ENHANCEMENTS FOR THE CAPS PROTOTYPING SYSTEM DESCRIPTION LANGUAGE SYNTAX-DIRECTED EDITOR

by

Scott Robert Grosenheider

March 1996

Thesis Advisors:

Shing
Luqi

Approved for public release; distribution is unlimited.
Enhancements for the CAPS Prototyping System Description Language Syntax-Directed Editor

Grosenheimer, Scott Robert

Naval Postgraduate School
Monterey, CA 93943-5000

Approved for public release; distribution is unlimited.

CAPS (Computer-Aided Prototyping System) is an integrated set of software tools that generate source programs directly from real-time requirements. CAPS users can specify the requirements of prototypes as augmented computational graphs using the graphics/text editor. The problem with the current version of CAPS is that most of the feasibility checks for the prototypes are currently enforced by the translator and the scheduler. Such an approach requires the engineers to go through the "edit, save file, then translate and schedule" cycle in order to find out if the control and timing constraints can be satisfied. The prototyping process can be made much more efficient and user-friendly if these checks are enforced by the CAPS PSDL (Prototype System Description Language) SDE (syntax-directed editor), where users can detect and receive warnings as they enter the design. This thesis focuses on the properties that must exist between processes and their inter-connected data flows in order for a prototype to be correct. It further modifies the PSDL SDE so that parts of the prototype are captured, combined, and manipulated in a way that provides the semantic information needed to determine if these properties have been violated. The new editor has been applied to several prototype examples. The results showed that, by catching errors during the editing phase, the user saves time, is better able to stay focused on the design, and is subsequently more productive.
ENHANCEMENTS FOR THE CAPS PROTOTYPING SYSTEM DESCRIPTION LANGUAGE SYNTAX-DIRECTED EDITOR

Scott Robert Grosenheider
Captain, United States Marine Corps
B.S., University of Nebraska, 1989

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

March 1996

Author: Scott Robert Grosenheider

Approved by:

Man-Tak Shing, Thesis Advisor

Luqi, Thesis Advisor

Ted Lewis, Chairman,
Department of Computer Science
ABSTRACT

CAPS (Computer-Aided Prototyping System) is an integrated set of software tools that generate source programs directly from real-time requirements. CAPS users can specify the requirements of prototypes as augmented computational graphs using the graphics/text editor. The problem with the current version of CAPS is that most of the feasibility checks for the prototypes are currently enforced by the translator and the scheduler. Such an approach requires the engineers to go through the “edit, save file, then translate and schedule” cycle in order to find out if the control and timing constraints can be satisfied.

The prototyping process can be made much more efficient and user-friendly if these checks are enforced by the CAPS PSDL (Prototype System Description Language) SDE (syntax-directed editor), where users can detect and receive warnings as they enter the design. This thesis focuses on the properties that must exist between processes and their inter-connected data flows in order for a prototype to be correct. It further modifies the PSDL SDE so that parts of the prototype are captured, combined, and manipulated in a way that provides the semantic information needed to determine if these properties have been violated.

The new editor has been applied to several prototype examples. The results showed that, by catching errors during the editing phase, the user saves time, is better able to stay focused on the design, and is subsequently more productive.
# TABLE OF CONTENTS

## I. INTRODUCTION

A. SOFTWARE AFFLICTION ............................................. 1

B. PROTOTYPING .......................................................... 3

## II. CAPS

A. PSDL ........................................................................... 8

   1. Operators ............................................................... 9

      a. Composite or Atomic ............................................ 9

      b. Time-Critical or Non Time-Critical ......................... 10

      c. Periodic or Sporadic ............................................. 10

   2. Streams ................................................................. 10

   3. Timers ...................................................................... 12

   4. Triggers ..................................................................... 12

   5. Conditionals .......................................................... 12

      a. Conditional Execution .......................................... 12

      b. Conditional Output .............................................. 12

   6. Exceptions ............................................................. 12

B. EDITORS ....................................................................... 13

   1. PSDL Editor ............................................................ 13

   2. Text Editor .............................................................. 13

   3. Interface Editor ....................................................... 14

   4. Requirements Editor ................................................. 14
5. Change Request Editor ........................................... 14

C. EXECUTION SUPPORT ........................................... 14
   1. Translator ....................................................... 14
   2. Scheduler ...................................................... 15
   3. Compiler ....................................................... 15

D. PROJECT CONTROL ............................................. 15
   1. Evolution Control System ................................. 15
   2. Merger ......................................................... 16

E. SOFTWARE BASE .................................................. 16

III. THE SYNTHESIZER GENERATOR .................................. 17
   A. BACKGROUND .................................................. 17
   B. USES .......................................................... 18
   C. BUILDING AN EDITOR ....................................... 19
      1. Abstract Syntax Rules ................................. 22
      2. Attribute Rules ......................................... 25
      3. Unparsing Rules ........................................ 35
      4. Transformation Rules ................................. 37
      5. Concrete Rules .......................................... 38

IV. SYNTAX-DIRECTED EDITOR (SDE) .............................. 43
   A. EXAMPLE OF SDE ........................................... 43
      1. Simple Tutorial ........................................ 43
      2. SDE Menu Functions ................................... 62
   B. NEW FUNCTIONALITY ....................................... 65
1. Scheduling Constraints ........................................ 65
   a. General ..................................................... 65
   b. Data Triggering Semantic Checking .................... 67
   c. Timing Constraint Semantic Checking ................. 69

C. MODIFICATIONS ............................................. 73
   1. Data Triggering Constraints ............................ 73
   2. Abstract Syntax Rule .................................... 74
   3. Attribute Rules ......................................... 77
   4. Unparsing Rules ......................................... 83
   5. Transformation Rules ................................... 84
   6. Concrete Rules .......................................... 84

D. TESTING ...................................................... 84

V. FUTURE RESEARCH AND DEVELOPMENT .................... 89

LIST OF REFERENCES ........................................... 91

APPENDIX A - PSDL GRAMMAR .................................. 93

APPENDIX B - ABSTRACT SYNTAX RULES ...................... 95

APPENDIX C - ATTRIBUTE RULES ............................ 103

APPENDIX D - AUXILIARY FUNCTIONS ...................... 119

APPENDIX E - UNPARSING RULES ............................. 241

APPENDIX F - TRANSFORMATION RULES .................... 249

APPENDIX G - CONCRETE SYNTAX RULES .................. 255

INITIAL DISTRIBUTION LIST ................................. 269
ACKNOWLEDGEMENT

To Dr. Shing, I would like to express my deepest gratitude for being so patient while explaining the required concepts over and over again. Your support, knowledge, and ability to grasp new things never ceases to amaze me.

To my children, Melessa and Kim, who have been known to call me “geek” a time or two in jest, I hope that some day you will be in my shoes. You have given a lot and given up a lot for me and I sincerely appreciate that. I know it is tough, especially for teenagers.

To my beloved wife, Joyce, who has always been there for me, I owe you everything. You mean everything to me and are the reason I have achieved as much as I have. You’ve given up so much in the past and expected so little in return. Words can’t describe my appreciation.

Finally, thank you God for making this all possible.
I. INTRODUCTION

Cheaper, faster, more reliable hardware has helped to make software the dominant factor in today's computing needs. While sky-rocketing demand for software continues, the requirements get much more demanding; especially so for real-time systems with their timing restrictions. Scheduling feasibility is one of the most important and demanding aspects of ensuring that the requirements of a real-time system are met because success means not only being correct, but also on time. Correctness refers to the precision and accuracy which must be adhered to. Being on time refers to the response time required of individual operations. CAPS (Computer-Aided Prototyping System) is an ongoing software engineering project at the Naval Postgraduate School that is targeted primarily at real-time systems. This thesis focuses on some of the issues related to whether or not the design of a real-time prototype is feasible and addresses checking some of these feasibility constraints earlier in the prototype development cycle. It modifies the Syntax-Directed Editor (SDE), which is one of the tools within CAPS, so that errors can be caught while still in the design phase, thereby saving the developer valuable time, enabling more accuracy in the design, and subsequently contributing toward higher productivity.

This chapter describes the current state of software development along with a partial justification for the need of supporting tools and prototyping. Chapter II gives a better description of CAPS, the tools it encompasses, and the steps required to create a prototype. Chapter III discusses the tool that is used to generate the SDE, the Synthesizer Generator. It summarizes the required specifications to creating an SDE. Chapter IV covers creating a short example prototype, some of the constraints that must hold for a real-time prototype to be schedulable, detailed changes that were required of the SDE to implement those constraints, and an example with the new editor. Chapter V summarizes with a conclusion and possible future enhancements.

A. SOFTWARE AFFLICTON

Most people who have had even a small amount of interaction with the development of software have probably heard of the term “software crisis”. Many will immediately begin to think of various problems that exist in the field which contribute to this ongoing situation. A
better term for this problem is “chronic affliction” [Ref. 1]. The word “crisis” tends to point to a single point in time where something dramatic happens, where “chronic” means recurring and “affliction” means anything causing pain or stress.

There are many problems in the field of software engineering and most of them are recurring and sometimes very painful. Probably the biggest problem facing software development today is the insatiable appetite for software that our world has amassed; an appetite that is constantly growing. The backlog of demanded systems continues to grow in numbers while the systems themselves are becoming much more complex.

Real-time systems are taking a bigger piece of this pie and among the most difficult to design right because of their restrictions on time. Many are critical in that failure can lead to great costs. History is already full of incidents caused by software failures such as the Bank of New York paying $5 million because of a software problem [Ref. 2] and the Patriot missile that missed a Scud missile because of a timing error and subsequently ended up in 28 Americans killed [Ref. 3]. Quality and user friendliness is expected in degrees rarely achieved to date. All this while many of the current projects are over budget, past due, and not meeting user expectations. Some projects are dropped after extensive amounts of resources have been invested. In fact, as many as 25% of the projects in large MIS organizations are never completed [Ref. 4]. All too often, cost estimates are only guesses because true requirements are simply not known. The bottom line is that the software engineering community must increase productivity dramatically to catch up, let alone keep up.

Probably one of the single most critical areas in which software engineers must get better at is capturing requirements. Most of the time, software products don't satisfy what the user really intended. It may meet the requirements specified by the user, but if it doesn't meet the users needs, its a bust. This is the major cause of unsatisfied customers.

Software engineers must get faster at putting out finished products. Too many times, developers fall behind and this is one of the biggest factors in cost over runs. One well documented fact is that errors get much more expensive in terms of time and money when discovered later in the development cycle. Making matters worse, requirements errors are not likely to be found until implementation. So then, it only stands to reason that the most expensive errors are the ones concerning missed, incorrectly, or inconsistently defined software requirements.
One of the best ways of ensuring that true requirements are captured, thereby enabling
the developer to give more accurate time and cost estimates and at the same time maintain a
much higher probability that the product will be good (as define by the user), on time, and on
budget, is with a technique called prototyping.

B. PROTOTYPING

Prototyping is an excellent tool for determining exactly what is needed by the system to
be developed. Its purpose is “to help customers understand and criticize proposed systems and to
explore the new possibilities that computer solutions can bring to their problems in a timely and
cost effective manner” [Ref. 5]. It is much like the architect who gets some guidance from the
client, goes away to draft up preliminary designs, and then comes back to review what has been
done. The architect knows that it may be all wrong; better to find out earlier when backtracking
is still reasonably cheap than later when it is extremely expensive. Once the designer better
understands what the requirements are, more meaningful estimates can be determined along
with better cost/benefit driven feasibility studies.

The best situation for prototyping is when there is uncertainty about what is required.
This could be uncertainty as to what exactly the customer desires, uncertainty about how to
make things happen, or both. In the former case, a client may not know exactly what it is they
want, certainly making it difficult for the developer. They may know what is needed but can't
describe it. Many times the client speaks in a whole different vocabulary because of her
expertise. They may actually or mistakenly have described exactly what is needed and end up
with a product that is totally unusable. The English language is very ambiguous at best, leaving
lots of room for error.

The second scenario where a software designer is unsure how to make things happen can
be a very difficult problem. The designer may not be sure of how to deliver what is wanted in
terms of functionality or performance. Understand data and control flow, functional processing,
or general understanding of the behavior of the system can be greatly enhanced through
prototyping. This situation exists when venturing into unfamiliar terrain and is almost always
the case when dealing with real-time systems. The timing constraints imposed can be
enormously complex. The client normally can't describe them while the analyst/designer has a
tough time trying to determine them [Ref. 3]. Sometimes whole designs must be redone in order
for it to be feasible in terms of scheduling. Many times it is hard to determine if requirements have been met, further justifying the use of prototypes for validation purposes.

While prototyping can be done merely to gather requirements (it is meant to be thrown away after requirements gathering is done), the focus is in terms of a development paradigm. That is the concept of prototyping in a evolutionary approach where the system goes through a continuous cycle of modification/demonstration/evaluation until it is right. An iterative approach that rapidly adjusts the behavior of the prototype based on feedback from the customer and designer. This allows incremental development vice a big bang approach. Most who have worked on a complex project will agree that it is easier to get it right by quickly and correctly doing a little at a time than by trying to do everything at once. The big bang approach gets even more unwieldy when the size dictates multiple designers. With each iteration, the customer and designer come to a more common understanding of what the customer really needs, regardless of preconceived notions by either party. CAPS endorses this evolutionary approach with software tools that support the designer in the task of building and executing prototypes.

One aspect of this process that makes it better than more traditional methods such as the waterfall method is it does not expect to be able to freeze the requirements. Realistically, requirements do not stabilize until the user gets some exposure to the system (especially with real-time systems). With most traditional methods, this means not having stable requirements until after implementation, thereby necessitating significant change during its maintenance phase. Changes that would not have been required had the system been right in the first place. With a prototyping approach, the user gets adequate exposure to the prototype to determine true requirements. And because it is an iterative process, and because the prototype is easier to modify than implemented code, the user will eventually be able to fine-tune tentative requirements into exactly what the system needs are.

This process results in higher quality implementation because fewer initial errors get by. That in turn, lowers maintenance costs, which is the dominant expense in larger systems, because fewer implementation errors means fewer discovered mistakes to correct for during maintenance (many maintenance changes are really errors), not to mention the fights over whose fault and who pays (requirements mistake vs.implementation error). Also, when changes need to be made, the prototype (which is more simple than the production code) can be manipulated/modified rather than the production code (which is more complex). Then when the required
changes (new requirements) are certain, the new implementation can occur. Much of the implementation may consist of old code but 1) it is a new clean implementation with clear, documented requirements (didn't do patchwork with old implementation) and 2) the changes are easier to achieve as the designer worked with a higher level abstraction.

Because constraints from both the problem domain and the programming domain together drive the requirements of a proposed system, both the user with his knowledge of the problem domain and the designer with her knowledge of the program domain must work together extensively. Prototyping helps create a common ground where they can communicate effectively by demonstrating the behavior of the proposed system. The disagreements that arise quickly uncover areas that need further thought/clarification.

Testing gets more mileage with the iterative approach. Most organizations do the bulk of their testing towards the end of the software development cycle. This means that when funds and time allocated for the development of the system are at the end, the organization is testing to see if the product answers the mail. The question that immediately comes to mind is, “what if it doesn't?” The answer is complex. Fixes can be minor or significant depending on how far off the mark the developer is and how much money, time, and patience the customer has. Most testing authorities will state that because testing is one of the most effective tools in developing correct software, it should be moved further up in the software development cycle. By introducing a prototyping methodology that has validation and verification built into the cycle, this problem can be avoided. Then, when final testing is done, it does not have to be as significant and there are no surprises. This further means that the first implementation will more reflect user needs. That, in turn, means that less modifications must be made later in the maintenance/evolution of the system thereby saving time and money and keeping the system more stable and useful longer.

In a study that looked at thirty-nine significant cases, it was determined that “rapid prototyping is indeed appropriate for large systems, and there seems to be more successful use of evolutionary prototyping than throwaway”, and further that “rapid prototyping has had a number of positive effects on both the software product and development process and that it can be used successfully in a variety of situations” [Ref. 6].
Figure 1 below, which was borrowed from the CAPS Tutorial [Ref. 7], helps to show the Prototyping Process. The shaded areas depict the cycle that is executed in order to get the requirements right before implementation.

![Diagram of the CAPS Prototyping Process]

Figure 1. The CAPS Prototyping Process.
II. CAPS

Prototyping can dramatically increase the effectiveness of building and supporting software systems. "A high level language, a systematic prototyping method, and an integrated set of computer-aided prototyping tools are important for realizing the potential benefits of prototyping" [Ref. 8]. The language is PSDL (Prototyping System Description Language) and the integrated set of computer-aided prototyping tools is CAPS (Computer Aided Prototyping System). Useful for any type of application, CAPS is primarily developed for real-time system prototypes and consists of software engineering tools which are linked together by a common interface. These tools assist the designer in constructing and executing the prototype [Ref. 8].

Several sources make reference to hard real-time systems which are those in which deadlines and requirements are guaranteed to be met under the worse-case situation. While some COTS (commercial off the shelf) products are available for real-time support, only CAPS generates code which satisfy the constraints of a hard real-time system [Ref. 3]. The CAPS development environment is displayed in Figure 2 below.

![Diagram of CAPS development environment]

**Figure 2. The CAPS Development Environment.**
A. PSDL

Most of the tools within CAPS operate on PSDL which is a prototyping language designed to enable a high level description of the system. PSDL is the means by which these tools communicate thereby assisting them in demonstrating the behavior of the prototype. It was designed to assist requirements analysis, feasibility studies, and in designing large embedded systems. Because this thesis involves modifications to the SDE, which is heavily intertwined with PSDL, more detail will be spent on PSDL and the PSDL Editor than other parts of CAPS.

An underlying computational model was chosen to make inter-component communications explicit [Ref. 9]. It is formally represented as an augmented hypergraph, G, where G=(V,E,T(v),C(v)) [Ref. 10]

where:
- V is a set of vertices (vertices represent operators),
- E is a set of edges (edges represent data streams),
- T(v) is the set of timing constraints for vertex v, and
- C(v) is the set of control constraints for vertex v.

One of the major strengths of CAPS is its support of the prototyping methodology. Prototyping loses much of its appeal when modifications to the prototype are as intensive as modification to production code. Because of this, PSDL was designed to help make the required design modifications, common to the iterative approach, as painless as possible.

Modularity, absolutely necessary to a good design, is supported by operators that explicitly communicate via data streams. A design is represented by a hierarchically structured network of these operators and data streams which are laid out as dataflow diagrams, enhanced with timing and control constraints. Dataflow diagrams convey a great deal of information about the internal structure of a process and yet are simple to read. Extending this diagramming technique with timing and control information made the technique much more powerful while maintaining its simplicity.

PSDL supports reuse by capturing attributes that describe both the interface and the behavior of components. The interface attributes captured include generics, inputs, outputs, states, exceptions, and timing information, while the behavior attributes are keywords, and formal and informal descriptions. These attributes can be used to retrieve reusable components and organize the software base.
Requirements tracing is supported in PSDL by a construct that links the requirements to the part of the prototype that implements it [Ref. 8]. This is important so that as the system evolves, obsolete or changed requirements can quickly be identified in the implementation and modified as necessary. It further helps ensure that the system and its documentation stay up to date.

PSDL further supports the requirements of a real-time system design: Control Constraints, which maintain preconditions on the firing of a module, filter a modules output, and/or control timers, and Timing Constraints, which implicitly determine when constrained operators will fire, are built into PSDL [Ref. 8]. They are discussed in more detail below. While PSDL was designed with a small set of constructs to make it very powerful, it was kept as simple as possible.

1. Operators

Operators can represent a function or state machine. The act of firing (executing) involves reading one data object from each input data stream and writing zero or one output to each output data stream. The operator represents a function if its output is solely dependent on the inputs to the operator; the same input will always produce the same output. If, on the other hand, its output depends on both inputs and values stored in its memory (internal state), it is a state machine. Operators can affect each other only by explicit data streams and can be further decomposed as design decisions and principles of abstraction dictate.

a. Composite or Atomic

Operators can be classified as composite or atomic. A composite operator is one that should be further decomposed. On the flip side, an atomic operator is one that should be decomposed no further. When analyzing an operator, the first question is, “Is there a component in the software base that will do this?” If the answer is yes, the newly designated atomic operator is implemented with that component. If, on the other hand, the answer is no, the next question is, “Does it make sense to decompose this operator?” If so, it is dubbed a composite operator and is decomposed. This will result in multiple operators that are each, in turn analyzed with the first question above. Of course, if further decomposition doesn’t make sense, the operator is atomic and an implementation must be created.

As with all data flow diagrams, the lowest levels incorporate atomic operators. The goal is to eventually have a significant number of those implementations via components
from the software base, which have a high degree of reliability.

b. Time-Critical or Non Time-Critical

The next classification of an operator is whether it is time-critical or not. Built into the definition of a real-time system is the understanding that there will exist operators that have a constraint on how long they have to complete execution. In PSDL, this is called, its maximum execution time (MET) and denotes the maximum amount of CPU time the operator can use for execution. If an operator is assigned a MET, it is obviously time-critical. If not, it is simply an operator like the ones we have all seen in normal non real-time systems. Time-critical operators can be broken down into two further categories.

c. Periodic or Sporadic

Again, only time-critical operators (has a MET) are further classified as to whether they are periodic or sporadic. A periodic operator has a regular interval in which it must fire and complete its execution. It is assigned a period (PER) which denotes the frequency in which the Scheduler makes a processor available for execution. Within that window, the operator must fire and complete execution. It is assigned a finish within (FW) time that denotes how long from the start of the window (PER) before execution must be completed. While the FW starts at the beginning of the PER, the MET doesn’t start until the operator fires which can be after the beginning of the PER.

A sporadic operator does not necessarily have a regular interval in which it fires and is triggered by the arrival of new data. It is assigned a maximum response time (MRT), which is the maximum amount of time between the arrival of new data on the input data stream(s) (which triggers the operator) and the time when the last output is put on the output data streams. In addition, it is assigned a minimum calling period (MCP). The MCP is a lower bound on the delay between two subsequent arrivals of triggering data on the input.

More discussion of both periodic and sporadic operators will ensue in chapter five when covering timing and control constraints.

2. Streams

Flows between operators can be data, control, or exception information. A stream is a communication link that connects two operators. The originator of the stream is called the producer operator while the user of the stream is called the consumer operator. A PSDL prototype is schedulable only if the graph is directed and contains no cycles. This is more
commonly referred to as a DAG (directed acyclic graph). When a cycle occurs, it indicates the presence of state information and must be dealt with.

*State streams* are the means by which state information and hence cycles in the graph are dealt with. The state stream provides a way of declaring and initializing the state. During scheduling, it is actually removed from the graph. Figure 3 shows how streams are depicted in the PSDL Editor of CAPS. Data streams can be broken down into two other classifications. The consumer operator determines what type of stream it is. If the consumer operator has a trigger of “Triggered By All”, then the stream is a *data flow stream* (triggering is discussed below). In all other cases, including “Triggered By Some”, it is a *sampled stream*.

![Diagram of a stream](image)

*Figure 3. Typical diagram of a stream. Data Streams look as pictured. State Streams have a bold arrowed line connecting the two operators.*

A data flow stream guarantees that no data is lost or duplicated [Ref. 10]. It is like a FIFO queue with a length of one. A consumer reads the data from its incoming data stream destructively so that it is no longer there. Underflow occurs if it attempts to read an empty stream. Overflow occurs if the producer operator tries to put new data to the stream before the consumer has read the old data.

Sampled streams do not guarantee against lost or duplicated data. They are like a single memory variable that can be updated or read zero or more times. In addition, reads are not destructive. The only way a data value will go away is if the producer replaces it with another.
3. Timers

_Timers_ are an abstract state machine that act like a stopwatch [Ref. 10]. Used for things such as time-outs or refresh rates, they keep track of the amount of time that elapses between certain events. The four primitive operations are READ, START, STOP, and RESET. These are absolutely essential in a real-time system.

4. Triggers

Any operator can have a trigger and there are two types; _BY ALL_ and _BY SOME_. For example, if an operator has “OPERATOR A TRIGGERED BY ALL X, Y”, it will fire when new data is on both X and Y data streams. X and Y may or may not be all of the input data streams. This guarantees the output is based on new input and can be used for synchronization purposes. If, on the other hand, it had “OPERATOR A TRIGGERED BY SOME X, Y”, it would fire when either data stream X or Y had new data on them.

5. Conditionals

There are two kinds of conditionals in PSDL which control the input and output of an operator. Conditionals can be combined with other constraints including triggers. While these could be implemented within the operator itself, this provides a quick, clear way of controlling the operator, assisting in the goal of making the prototype easier to modify.

   a. Conditional Execution

   Conditional execution, sometimes called _execution guards_, enforce a pre-condition before allowing the operator to fire. They entail the keyword “IF”. The following example shows that the operator will not fire until the condition Brake_On is true: “OPERATOR Suspend_Cruise_Control TRIGGERED IF Brake_On”.

   b. Conditional Output

   Conditional outputs, sometimes called _output guards_, determine whether or not data is written to the output data stream. It does not, however, have any control over the firing of the operator. The condition can depend on operator input/output and timer values.

6. Exceptions

PSDL has a built in abstract data type called exception. It can be used to create exceptions with a given name, detect whether a value is an exception, and determine whether a value is “normal” (a keyword in PSDL). As mentioned previously, they can be transmitted over data streams.
B. EDITORS

There are several editors within CAPS providing a range of functionality. From basic editing of text files to building prototypes. These options/resources fall under the “Edit” pull-down menu.

1. PSDL Editor

A prototype is built with the PSDL Editor. It is composed of three parts: the Syntax Directed Editor, the Graph Viewer, and the Graphic Editor. They allow the designer to create the CAPS data flow diagram and PSDL program, assigning all timing and control constraints necessary to ensure the proper design of the prototype and its components.

The Graphic Editor is used to build the data flow diagram and to specify some of the timing constraints. This provides a way to show the design structure in a simple diagrammatic way. A great deal of information can be represented in somewhat simple diagrams and modification is quick and simple.

The Syntax-Directed Editor (SDE) captures the information already entered into the Graphic Editor. Further, it allows for entering PSDL descriptions that are free of syntax errors by immediately notifying the user when they arise. It is this functionality that is extended by this thesis. The restrictions imposed by the constraints have an impact on whether the prototype is schedulable. Chapter IV covers these restrictions and some of this information has been incorporated in the SDE so that design errors related to timing and control constraints, in addition to syntax errors, are discovered while still in this early phase of design. Chapter III contains an example on creating a prototype.

If, while in SDE, the current position of the prototype pertains to the data flow diagram, the Graph Viewer displays that view of the data flow diagram. This helps keep the designer keyed into exactly what part of the prototype s/he is currently in and provides an outstanding overview of what process is currently under review.

2. Text Editor

The text editor can be one of several text editors depending on which one is chosen as the default. CAPS provides a convenient interface to the editor chosen. The user can select which editor is desired by choosing “CAPS Defaults”. The possibilities are vi, emacs, and the Verdix Ada Syntax Directed Editor.
3. Interface Editor

Under the "Interface" option, CAPS provides a seamless interface to TAE+, a versatile tool for creating and manipulating dynamic window-based user interfaces. When done, skeleton code is generated, contributing to the goal of generating the prototype quickly and efficiently. TAE+ allows for code to be generated into one file or into multiple files. While generating code under CAPS, the single file option is normally chosen (can be multiple files), and the code is placed in the prototype directory called: <prototype name>.RAW_TAE_INTERFACE.a.

4. Requirements Editor

While the goal is to have a tool that allows mapping from the requirements to the portions of the prototype that implement it, at current the "Requirements" option simply provides a window that lists the files with the extension "req". From there, any one file can be selected at which time the default editor is utilized to make necessary changes.

5. Change Request Editor

Like the requirements editor, the "Change Request" option brings up a list of files with a specified suffix; this time "cr". Again, the user picks one and the default editor is summoned to edit that file. The hope is for this option to call a sophisticated change request tracking/editing tool.

C. EXECUTION SUPPORT

Rapidly constructing and updating a prototype depends on efficient execution support. These options/resources fall under the "ExecSupport" pull-down menu.

1. Translator

The Translator "augments the implementations of the atomic operators and types with code realizing the data streams and activation conditions, resulting in a program in the underlying programming language that can be compiled and executed." [Ref. 10]. Essentially, it generates code that binds the components extracted from the software base or custom built, depending on whether or not they are in the library.

With Ada as the implementation language, it translates PSDL code into Ada wrapper code to realize the control constraints and instantiates Ada tasks for PSDL data streams. It expects a complete PSDL program as input, and creates several packages which make up, in part, the supervisor module of the prototype.
2. Scheduler

Because real-time systems have constraints and finite resources that tend to introduce dependencies between the functions of the system that would otherwise be independent, small changes can significantly impact the design [Ref. 3]. And because scheduling is a major factor in correctly designing a real-time system, that step must be automated so that excessive amounts of time are not spent on scheduling with every small change to the design.

The Scheduler generates two schedules; a high priority Static Schedule and a low priority Dynamic Schedule. The former allocates time slots for the time critical operators and if successful (the prototype is schedulable), all operators are guaranteed to meet their required timing constraints. If the prototype is not schedulable, the scheduler gives some diagnostic information that can help the designer see why.

The latter invokes the non-constrained operators during execution with the time slots not previously allocated. Translation is required before scheduling which is required before compilation.

3. Compiler

The Compiler option interfaces with the Sun Ada compiler. The prototype must have been successfully translated and scheduled prior to compilation.

D. PROJECT CONTROL

1. Evolution Control System

As the size of a project gets bigger, the number of people working on that project grows. This means more time is spent on communications and less time is spent on analyzing/designing, etc. Large projects also dictate a longer time until completion which means more turnover of personnel. Both of these problems can be minimized by ensuring that all documentation is stored and managed on-line.

The Evolution Control System (ECS) is designed to give automated help to the difficult task of coordinating concurrent efforts of prototype design team(s) and managing the multiple design versions that can be produced. The prototype development data is stored in a design database (DDB) for persistent storage.
2. Merger

The Merger helps to combine the product of two or more independently developed prototype changes thereby facilitating parallel enhancements and applying common changes to multiple versions. It warns of possible conflicts in the merging of two changes and when none exists, will create a PSDL program for the newly created prototype.

E. SOFTWARE BASE

Accessible through the “Databases” pull-down menu, the software base is currently designed to provide both “Browse” and “Query” capabilities for accessing a repository of reusable Ada and PSDL components. The user can browse by either types or operators and can query by keyword or PSDL specification.

Standards on how to specify a reusable component are not yet in place. Because of this and the difficult nature of the task, tools for finding and adapting appropriate software components have yet to live up to expectations for strong code reuse. Currently based on parameters, work on this part of CAPS is ongoing to provide better underlying matching capabilities.
III. THE SYNTHESIZER GENERATOR

Attribute Grammars have been used extensively in building compilers with their ability to accomplish translations and specification of static semantic analysis. Attribute Grammars have been used in porting code from legacy systems to newer languages. In general, they have been used in the development of many tools of modern day software engineering in an attempt to provide products that enhance quality and productivity. One of these tools is the Synthesizer Generator which is used to generate the Syntax-Directed Editor (SDE) within the PSDL Editor. A SDE, language-based editor, or smart editor, is an editor tailored to a specific language (in this case, PSDL), utilizing the grammar, structure, and static semantics of that language to assist the user in writing correct programs.

A. BACKGROUND

The Synthesizer Generator (SynGen) is a system for developing smart editors which use the knowledge of the language itself to achieve specialization. It automates the implementation of a desired language based editing environment. The knowledge of the language enables the editor, depending on how it is built, to provide feedback to the user concerning whether a program contains syntactic or semantic errors, where they are and recommendations on how to fix them. Inconsistencies can be identified, along with other types of analysis. It can be used for conversions, translations, and transformations. Editors produced can also control how the user proceeds in many various situations. It creates the language-specific editor from a specification of the language’s abstract syntax (abstract syntax rules), context-sensitive relationships (Attribute Rules), display format (Unparsing Rules), concrete input syntax (Concrete Rules), and transformation rules (Transformation Rules) and performs analysis, translation, and error reporting with the use of an immediate-computation paradigm [Ref. 13]. The rules identified in parenthesis will be covered individually in order to give a clear representation of the required specifications for a SDE. The immediate-computation paradigm simply means that all attributes are validated with each and every update of the program.

Every object within the program, including the whole program itself, is represented as a consistently attributed derivation tree that goes through many transformations as the program is edited. The rules specified according to the language are used to check these attributes with
every program modification so that the integrity of the tree stays in tact. The language used to
do the specifications is called the *Synthesizer Specification Language* (SSL), which is built upon
a foundation of attribute grammar, a type definition facility, and the application domain of
language-based editors [Ref. 13].

B. USES

The editor created can be a hybrid of many different types of more specialized editors/
tools. Understanding this, those specialized editors and tools should discussed lightly.

*Structure editors* view a program as a hierarchical composition of individual structures.
This means that any component, including the whole program can be broken down into its
components. These structures, also called *templates* are predefined formatted patterns
representing the constructs in the language; a For Loop for example. Seeing a program this
way leads to constructing it by inserting templates into placeholders. For example, if you wanted
to insert a FOR LOOP inside an existing IF statement, you would highlight the placeholder
inside of the IF statement and insert the FOR LOOP as displayed in Figure 4. If the FOR LOOP
template was highlighted and deleted, the old placeholder would return. Obviously all template
insertions are controlled and therefore ensure correct syntax.

![Figure 4. Sample display of using templates.](image)

The typical *text editor* allows for character-oriented and line-oriented textual operations.
This is the type of editor that most of us are used to and can be very quick when the user is very
familiar with the syntax of the language. SynGen allows for both forms of text operations.
WYSIWYG editors allow for specialized viewing of the information that is stored. They can, as the name implies, show you what you have. They can also display information in a number of ways that can benefit the user; for example, some forms of display are more informative. By allowing for the control of what is displayed, SynGen can achieve variations of WYSIWYG.

Spreadsheet applications are useful in that whenever an update is made in the application, changes are automatically made throughout. In SynGen, this aspect is captured with its immediate-computation paradigm previously mentioned. It enables the editor to complete the analysis, error message generation, and code generation with each incremental change. This is a very important aspect as many types of information can be stored and updated with each modification. Executable code can be output with each change, allowing for immediate feedback on changes made. This becomes a significant productivity enhancement when going through the “modify, run, evaluate” cycle of testing. Correctness and proofs can be supported/maintained while editing if the rules are spelled out in the SSL.

Incremental code generation was just mentioned above. But what about generating code based on knowledge of another language and a program in the language. This is a hot topic with projects that seek to re-engineer existing systems. There are many legacy systems that have little-to-no documentation. This lack of documentation makes software engineers very reluctant to tackle these systems when it comes to re-writing in newer languages. SynGen can help significantly with this difficult task.

C. BUILDING AN EDITOR

The objective of this chapter is to familiarize the reader with the SynGen and how it works. For a full and in-depth analysis of the SynGen, see references [Ref. 13, 15]. The ideal editor will normally have a combination of all the features discussed above making it a hybrid editor. Building an editor requires that five different specifications be made in SSL. As these specifications are outline, also covered will be terminology of the SynGen. The five specifications, mentioned earlier, are abstract syntax, context-sensitive relationships, display format, concrete input syntax, and transformation rules. Before doing that however, it might be more useful to first define a small language to help with explanations. Look at Figure 5, which is the small subset of PSDL used in designing the initial SDE for CAPS [Ref. 14]. This
subset should be small enough to provide a good example and large enough to get the reader a little accustomed to PSDL. The complete PSDL grammar is in Appendix A.

```
psdl
  = (component)
component
  = data_type | operator
data_type
  = "type" id type_spec
operator
  = "operator" id operator_spec
type_spec
  = "specification" [typeDecl] "end"
operator_spec
  = "specification" (interface) "end"
interface
  = attribute [regmts_trace]
attribute
  = input | output
input
  = "input" typeDecl
output
  = "output" typeDecl
regmts_trace
  = "by requirements" id_list
typeDecl
  = id_list ":=" id
id_list
  = id ("," id)
id
  = letter (alphanumeric)
alphanumeric
  = letter | digit
letter
  = "a..z" | "A..Z" | "_"
digit
  = "0..9"
```

**Figure 5. Subset of PSDL Grammar.**

This may look somewhat confusing at first, but is really rather simple after a few minutes worth of inspection. Brace brackets ({}) indicate zero or more iterations, while square brackets ([]) indicate zero or one iteration. The pipe (|) and a small circle with a plus sign in it (used in the next figure) indicates exclusive-or. One might read this as, a psdl prototype is composed of zero or more components, which are each either a data_type or an operator. A data_type is composed of the literal "type", an id, and a type_spec, and so on.

Figure 6 depicts the same information in a form that may be more useful. The only true terminal nodes on this tree are the ones surrounded in quotes. Other nodes that are leafs are further expanded somewhere else in the tree. The thing to remember is that much like a compiler, SynGen will search the tree until it correctly reaches a terminal node ensuring that the pieces of the program match the syntactic constraints of the language.
Figure 6. Subset of PSDL Grammar.
1. Abstract Syntax Rules

The abstract syntax is the core of the editor and is defined as a set of grammar rules. Anything constructed or modified within the editor will be represented by a derivation tree that is built based on the grammar. It maintains how legal tokens and productions are allowed. Figure 7 shows one of several possibilities for representing the abstract rules for the grammar previously defined. Not all of the productions were specified to keep its size more manageable.

```
root prototype:
/***** PARTIAL PSDL LEXEMES *****/
BOOLKW
: <"BOOLEAN" > |
<"boolean" >;
TRUEKW
: <"TRUE" > |
<"true" >;
FALSEKW
: <"FALSE" > |
<"false" >;
IDENTIFIER
: IdentLex[a-zA-Z][a-zA-Z_0-9]* >;
/***** PARTIAL PSDL PRODUCTIONS *****/
prototype
: Prot(psdl_components);
list psdl_components;
psdl_components
: PslNil() |
PslPair(component psdl_components);
component
: ComponentNull() |
Data(id type_spec) |
Op(id operator_spec);
operator_spec
: OpSpec(interface_list);
type_spec
: TypeSpec(optional_type_declaration);
optional list optional_interface;
interface_list
: InterfaceNil() |
InterfaceList(interface interface_list);
interface
: InterfaceNull() |
Interface(attribute optional_requirements);
attribute
: Input(type_decl) |
Output(type_decl);
optional optional_requirements;
optional_requirements
: OptRequstsTraceNull() |
OptRequstsTracePrompt() |
OptRequstsTrace(id_list);
type_decl
: TypeDecl(id_list id);
optional optional_type_declaration;
optional_type_declaration
: OptTypeDeclNull() |
OptTypeDecl(id_list id);
list id_list;
id_list
: IdNil() |
IdPair(id id_list);
id
: IdNil() |
Id(USDENTIFIER);
```

Figure 7. Abstract Syntax Declarations.
Before proceeding, some of SynGen's terminology should be explored. These first three definitions are recursively defined and will make a beginner's head hurt until s/he has gone through several examples. A phylum is a set of terms. A term is the result of applying a k-ary operator to k terms of the appropriate phyla (plural of phylum). A k-ary operator is a constructor-function mapping k terms to a term [Ref. 15]. The phylum associated with a non-terminal is the set of derivation trees that can be derived from it. Those derivation trees (called terms) are derived by going through the productions identified by the operators. We will see an example of this shortly. Phylum declarations are either productions or lexemes. The legal production declarations allowed, which are described in the abstract syntax rules, take on a form something like:

```plaintext
phylum-name : operator_name (phylum1, phylum2, ..., phylum_k);
```

where:

- phylum_name is the particular phylum this production applies to,
- operator_name is any legal identifier that refers to a particular production, and
- phylum_i represents a non terminal of the grammar.

The legal tokens allowed, defined by the lexeme declarations within the abstract syntax rules, take on a form such as:

```plaintext
phylum-name : lexeme_name < regular_expression >;
```

where:

- phylum_name is the particular phylum this lexeme belongs to,
- lexeme_name is used in the definition of the concrete input grammar, and
- regular_expression is the description of the token.

A few more definitions are required at this point. Each phylum contains a unique term called its completing term and placeholder term. While the same term can be both, there are differences between them that must be discussed. The completing term is used to construct the derivation tree's default representation. Whenever there is an unexpanded occurrence of a phylum in the derivation tree, there will be an instance of the appropriate completing term. The placeholder term identifies where subterms can be inserted or swapped. This will become more apparent when the transformation and unparsing rules are covered. The first operator declared for a phylum is that phylum's completing operator. The completing operator is used to build the completing term; it is always the first operator in the completing term. The rest of the
completing term depends on whether it is ordinary, a list, or optional.

With ordinary phyla (not list nor optional), both completing and placeholder terms are created by applying the completing operator to the completing terms of its arguments. Nullary terms are stopping points and their parenthesis are optional. For example, the completing term for type_name is TypeName(IdNull). The completing operator of type_name is TypeName, and it has only one argument. The completing operator of its argument, id, is IdNull, which is a nullary operator. See Figure 8 below to show the components affected.

![Figure 8. Components of a Completing Term.](image)

If the phylum is of non-optinal list phyla, the completing term and placeholder term are also equal and are built by applying its binary operator to the completing term of its left argument phyla and to the list’s nullary operator [Ref. 13], resulting in a singleton list. This makes more sense when it is realized that all list phyla must have two argument phyla, where the first is another phylum and the second is the list itself (all lists are right recursive). For example, the completing term for id_list is IdPair(IdNull, IdNil). The binary operator is “IdPair”, the completing term of its left argument is “IdNull”, and the list’s nullary operator is “IdNil”. A look back to Figure 7 will help follow this reasoning.

Attempting a combination involves using both rules. For example, the completing term of type_decl is TypeDecl(IdPair(IdNull, IdNil), TypeName(IdNull)). Noting that type_decl is ordinary, you combine the completing operator, TypeDecl, to the completing terms of its arguments; id_list and type_name. Since their completing terms are already known (from the last two paragraphs), it now just a matter of combining them.

When dealing with optional phyla, the completing term and placeholder term are different. For optional, non-list phyla, the completing term is built from its first nullary operator
while the placeholder term is built from the first operator after the placeholder term. For example, the completing term of optionalRequirements is OptReqmtsTraceNull. The placeholder term is OptReqmtsTracePrompt. These two nullary terms are in the optional phyla because the first will result in nothing being displayed in the editor while the second will allow the editor to display a prompt. This is covered in the unparsing rules.

With the last category, optional list phyla, the completing term is the nullary operator and the placeholder is the same as for the non-optional list phyla covered above.

2. Attribute Rules

So far, the underlying structure of the editor has been described. The next question that comes to mind is how all of the syntactic and semantic checking/analysis is done. This involves various parts of the derivation tree knowing about and having some sort of an understanding of other parts of the tree. In other words, information must be passed up/down and back/forth within the derivation tree. It is accomplished with attributes and attribute equations.

The attribute rules make up an attribute grammar, which is a context-free grammar (CFG) that is extended by the use of attributes which are attached to non-terminals and defined by attribute equations [Ref. 13]. There are several kinds of attributes available to SynGen. While all are covered in the references [Ref. 13, 15], only three will be discussed here as they apply directly to the SDE within CAPS.

The editor designer can use local attributes, which are associated with a particular production. They are declared with the reserved word, “local” and are declared with the production attribute equations. The format of a local attribute declaration is as follows.

```plaintext
local attribute_type attribute_name;
```

where:

- `local` is a keyword,
- `attribute_type` is any predefined type,
- `attribute_name` is any valid identifier.

The attributes that are associated with phyla, instead of productions, are broken into two mutually exclusive (disjoint) sets: synthesized attributes and inherited attributes.

*Synthesized attributes* are attributes that are built up. They are attributes whose values are propagated to the left in attribute equations and up the attributed tree. An example will be given in a moment.
Inherited attributes are attributes that are passed down. They are attributes whose values are propagated to the right in attribute equations and down the attributed tree. The attribute declaration format is as follows.

```
  a_phylum { synthesized attribute_phylum attribute_name;
               inherited attribute_phylum attribute_name;};
```

where:

- `a_phylum` is the phylum or phyla (comma separated) that will have the attribute,
- `synthesized` is the keyword to declare a synthesized attribute (syn is OK),
- `inherited` is the keyword to declare an inherited attribute (inh is OK),
- `attribute_phylum` is the attributes type and can be built-in or used defined,
- `attribute_name` is any valid identifier.

It should also be mentioned that the two attributes can be in either order and can be in separate declarations if desired.

Attribute equations, which are used to assign values to attributes and evaluate values of attributes, take on a slightly different form depending on if they are for synthesized or inherited attributes. The synthesized attribute equation and inherited attribute equation respectively, are as follows.

```
  phylum_name.attribute_name = subordinate_phylum_value;
  subordinate_phylum_value = phylum_name.attribute_name;
```

where:

- `phylum_name` is the phylum that owns the attribute,
- `attribute_name` is the attribute owned by phylum_name,
- `subordinate_phylum_value` is a little bit tricky. It can be any value consistent with the attribute’s type including another attribute, the same attribute from another phylum, or a function call, and is associated with a lower level of the attributed tree.

In some abstract syntax declarations (a list for example), the `phylum_name` is mentioned more than once. Because of this, additional notation must be introduced. The `phylum_name` by itself implies its first occurrence. Another way to depict this is with $$$. The second occurrence, would be indicated as `phylum_name$$2`. Then, every subsequent occurrence gets the number after the $ symbol incremented by one.
As an example, review the abstract syntax presented in Figure 7 above and notice that a prototype is simply a list of components. Recall that all lists are binary and right recursive. A component is either a data_type denoted with the phylums id and type_spec, an operator with the phylums id and operator_spec, or null. The id simply identifies the name of that component, whether it be a type or operator. Obviously, we do not want two components to have the same name, regardless of whether they are types or operator.

In the PSDL Editor, the graphic editor allows the designer to draw and name operators. This information is propagated to the prototype when exiting back to the SDE. That means the operators are already defined when you enter the SDE. It might be beneficial to notify the designer when trying to assign a new data_type with the same name as an existing operator. After the designer enters three operators (Op1, Op2, Op3) in the graphic editor and returns to SDE, the attributed tree would look similar to Figure 9 (attributes are bold typed).

Figure 9. Partial Derivation Tree after returning from Graphic Editor.
The information about operator names must be sent to where data_type names are declared. Another way of saying this is that operator id attributes must be synthesized and then inherited by the data_type. Currently, there are only three components, and all three are operators. When data_types are added, the list of psdl_components will grow containing both operators and data_types.

The first thing that must be done is to collect up all the operator id’s. Because they exist at the id phylum, that is where we will put the synthesized attribute declaration. That attribute will be passed up to the component phylum so it too will need an id attribute. And because they will be collected up into a set at the psdl_component level and subsequently passed to the prototype level, we must define a structure to hold a set of id’s and declare attributes of that type at the psdl_component phylum and the prototype phylum. Of course the skeleton attribute grammar must be in place for this to be inserted into. It looks like the production portion of the abstract syntax declarations displayed earlier. See Figure 10 to understand what is required in collecting all operator id’s at the prototype phylum. All additions to the abstract syntax are in bold type.
Figure 10. Partial Abstract Grammar showing Synthesized Attributes.

When studying the changes made for attribution, notice the way upper and lower case is used. This is very important to help keep from getting confused when looking at larger specifications such as that for psdl.

Figure 9 was duplicated in Figure 11 below except that the newly declared synthesized attributes are attached to the appropriate phylum.
Figure 11. Partial Derivation Tree reflecting Synthesized Attributes.

Now we have all the operator id’s at the prototype phylum. The next thing to do is to send that list of operator names back down to the component phyla so that whenever a data_type is about to be identified, the editor can make sure the id doesn’t already exist. The nullary phyla do not have need of it, as they have nothing to compare it against. It is much simpler and is essentially the same process, only in reverse; it is inheritance, and is shown in Figure 12. The only possible tricky part is when dealing with lists. At phylum psdl_components, the list must be passed down to the current component and to the rest of the list (psdl_components$2).
The problem now is that the component has the list but doesn’t know what to do with it. This answer to this involves the use of local attributes, rules defining error attributes, and possibly auxiliary functions. The functions for defining error attributes and auxiliary functions can be with the attribute rules or within another file (more common).

First, declare a local BOOL (predefined in SSL) attribute called duplicate_id_error within the component phylum under the Data operator. This will be set to true if the current data_type id already exists within the inh_op_id_set. Then a function must be written to search the list for the data_type id. Figure 13 shows where this would be added.
Figure 13. Partial Abstract Grammar showing Inherited Attributes.

Most of SSL including the auxiliary functions that can be written are very much like the C language. The "!" symbol, for example, means Not and the "(expression) ? :" is a conditional expression. Figure 14 below shows what the Id_In_Op_Id_Set would look like as an auxiliary function. Immediately following that is Figure 15 which represents the newly declared inherited attributes attached to the appropriate phylum.
In Figure 16, all of the attribute modifications are reflected at once. There are no big surprises except for combining synthesized and inherited attributes in the same declaration. The next section will concentrate on what the user sees, including notification of an error identified in the attribute rules.
Figure 16. Partial Abstract Grammar Showing All Attributes.
3. Unparsing Rules

As just mentioned, the editor can now determine if the user is attempting to assign an operator name to a `data_type` that already exists. In fact, there is a local attribute that contains the message to be displayed to the user if this situation arises. It is up to the unparsing rules to display the string defined in the attribute rules. The unparsing rules define everything that the user sees. They also controls which nodes of the abstract syntax tree are selectable and which productions are editable. The form in which unparsing rules appear is as follows.

```
phylum : operator [ unparsing_syntax ];
```

where:

- `operator` and `phylum` match the abstract syntax one for one, and the
- `unparsing_syntax` is the specification for how that part of the tree is displayed and is broken up into a left side and a right side, coinciding with the productions of the abstract syntax tree. The left and right side are divided by a `"::="`, denoting immutable text, or a `"::="`, denoting mutable text.

There is another symbol called the `selection symbol` which represents the position of each phylum occurrence. Which one is used depends on whether or not that phylum occurrence should be a resting place for the editor. The `"@"` selection symbol specifies that the phylum is a resting place while the `"^"` selection symbol specifies that it is not. The designer of an editor must keep in mind that almost all phyla exist once on the right hand side (RHS) and once on the left hand side (LHS). Therefore, if a phylum is or is not to be selectable, both occurrences must be considered. It makes sense to use one side or the other to determine selectability. The examples here will use the LHS, which means the selection symbols on the RHS will be `"^"`.

Control characters are allowed to help control the display of the screen. Table 1 lists them, while only three will be used here; `%t` for tab, `%b` for back tab, and `%n` for newline.
Table 1. SSL Display Formatting Commands.

The unparsing rules for the PSDL subset presented earlier are shown in Figure 17. Notice that whether a phylum is selectable is determined by its LHS occurrence. Also notice some consistency was maintained with respect to placement of formatting commands; whether at the beginning of the appropriate line or the end of the preceding line.

All of keywords or phrases that were identified in the PSDL grammar seemed to disappear in the other rules. Here, however, they have returned and are displayed in the appropriate places.

One last thing to note is the specification for a data_type where the user is now notified about “duplicate_id_msg” errors. The string variable, “duplicate_id_msg”, is always displayed at the Data production. The difference is that, if an error exists, the string variable will contain a message stating that fact, and if an error does not exist, the variable string will be a nullstring.
4. Transformation Rules

Transformation rules allow for the restructuring of objects. While transformations can include various kinds of computations, this section will focus on the more general case of template insertion. This can only happen when the selection is a placeholder. The format of a typical template insertion is as follows.

```plaintext
transform phylum on transformation_name pattern : expression;
```

where:

transform is a keyword identifying a transformation,
phylum identifies the phylum that this transformation can be done on,
on is another keyword,

transformation_name is the descriptive identifier in the help window which the user selects with the left mouse to activate the transformation,
pattern is either the same as the phylum or some appropriate sub-phylum, and
description is the template that is inserted where the pattern was.
The transformation rules for the PSDL subset is displayed in Figure 18.

```
transform component
on "type"
  <component> : Data{id},<type_spec>),
on "operator"
  <component> : Op{id},<operator_spec>);
transform attribute
on "input"
  <attribute> : Input<input>,<optional_requirements>),
on "output"
  <attribute> : Output=output>,<optional_requirements>);
transform optional_requirements
on "enter_requirements"
  <optional_requirements> : ReqmtsTrace{id_list>);
transform optional_type_declaration
on "enter_declaration"
  <optional_type_declaration> : OptTypeDecl{id_list>,<type_name>);
```

**Figure 18. Transformation Rules for Abstract Syntax.**

5. Concrete Rules

The concrete rules, or concrete input syntax, is needed to provide the editor with the
ability to do text input and to load a pre-existing program into the editor.

It allows for text editing by parsing and then translating the string to the corresponding
term of the abstract syntax. While the subterm is being edited, it can be any string. Once the user
attempts to move to a different part of the structure, it is parsed. If syntactically correct, it
replaces the term in the abstract syntax. If it is not correct, the cursor is positioned at the
beginning of the string and an error message is displayed.

It allows for editing existing programs by specifying rules that build a tree structure
coinciding with the abstract syntax rules previously defined. This means that as a minimum, the
concrete rules must specify a phylum in the input syntax to be associated with all phylum of the
abstract syntax which are to be edited as text. If pre-existing programs are to be re-edited, the
whole language must be supported.
There are several parts to the concrete rules, most of which is similar to what has already been covered above. The rules include attribute declarations which have a form such as:

\[
\text{phylum\_name \ (type\_attribute \ attribute\_phylum \ attribute\_name)};
\]

where:

- \(\text{phylum\_name}\) is a phylum within the input grammar,
- \(\text{type\_attribute}\) is either synthesized or inherited,
- \(\text{attribute\_phylum}\) is the attributes type and can be built-in or used defined,
- \(\text{attribute\_name}\) is any valid identifier.

*Entry declarations* are required to create a correspondence between the entry points within the input syntax and the appropriate selection within the abstract syntax. Their format is as follows:

\[
p \sim P.t;
\]

where:

- \(p\) is a phylum such that the current selected component is of that phylum, and
- \(P\) is an input phylum such that the input is parsed to see if it is of that phylum, and
- \(t\) is a synthesized attribute that if the input is of phylum \(P\), is used to replace the current selected component [Ref. 13]. In other words, if the two phylums match, the attribute goes in the derivation tree.

The input syntax must have tokens, so there are also lexemes in the concrete rules. There is one rule for each keyword or token. All white space is ignored during the parsing of the input program. The format of a lexical phylum declaration is the same as for the abstract syntax rule. For convenience, it is repeated below.

\[
\text{phylum\_name : lexeme\_name \ regular\_expression} ;
\]

Last, but certainly not least, is the productions of the input grammar or *parsing declarations.* A slight difference from what was explained previously is the use of the \$\). In this context, the phylum appended with \$1\) means the first occurrence and \$2\) is the second occurrence, and so on. The productions of the input syntax can be ambiguously specified in which case there must be disambiguating precedence rules. The general form is shown next.

\[
\text{phylum\_name ::= (phylum\_token\_comb) \ (LHS = RHS);}
\]

where:

- \(\text{phylum\_name}\) is self explanatory,
:= is the symbol to separate the LHS phylum name from the RHS symbols, phylum_token_comb is a phylum, token, or a combination of both which is parsed,

LHS=RHS is the assignment of some value to the attribute of the phylum_name.

On the following page is Figure 19, which contains the concrete rules for the subset of PSDL that has been the ongoing base throughout this chapter. The use of inherited attributes was not covered in this section as it is somewhat more detailed. It is covered extensively in references [Ref. 13, 15], however, for the interested reader. Also, the concrete rules for PSDL in the CAPS SDE do use inherited attributes to deal with the fact that the productions build lists in reverse order.

Hopefully several things have been accomplished in this chapter. First, the reader should have a better appreciation for how powerful the SynGen really is. Second, s/he should have a little better grasp of what PSDL is and how it is structured. In the next chapter, a small sample application is introduced. In that section, particularly when dealing directly with SDE, the reader should be continually reminded of the rules discussed in this chapter. Significant aspects of the first four rules are in that sample application. Everything shown to the user was specified in the unparsing rules. All warning and error messages are defined and discovered within the attribute rules and auxiliary functions. All transformations in the help window (bottom) of the SDE are specified in the transformation rules. While not evident in the sample application, bringing an existing prototype into the SDE requires the concrete rules. And finally, none of the above would be possible without the derivation tree representation of the prototype itself, specified through the abstract syntax rules.
/* Attribute Declarations */

PROTOTYPE
{synthesized prototype t;}

PSDL_COMPONENTS
{synthesized pSDL_components t;}

COMPONENT
{synthesized component t;}

OPERATOR_SPEC
{synthesized operator_spec t;}

TYPE_SPEC
{synthesized type_spec t;}

OPTIONAL_INTERFACE_LIST
{synthesized optional_interface_list t;}

INTERFACE
{synthesized interface t;}

ATTRIBUTE
{synthesized attribute t;}

OPTIONAL_REQUIREMENTS
{synthesized optional_requirements t;}

TYPE_DECL
{synthesized type_decl t;}

OPTIONAL_TYPE_DECLARATION
{synthesized optional_type_declaration t;}

ID_LIST
{synthesized id_list t;}

ID
{synthesized id t;}

/* Entry Declarations */

prototype
~ PROTOTYPE.t;

psdl_components
~ PSDL_COMPONENTS.t;

component
~ COMPONENT.t;

operator_spec
~ OPERATOR_SPEC.t;

type_spec
~ TYPE_SPEC.t;

optional_interface_list
~ OPTIONAL_INTERFACE_LIST.t;

interface
~ INTERFACE.t;

attribute
~ ATTRIBUTE.t;

optional_requirements
~ OPTIONAL_REQUIREMENTS.t;

type_decl
~ TYPE_DECL.t;

optional_type_declaration
~ OPTIONAL_TYPE_DECLARATION.t;

id_list
~ ID_LIST.t;

id
~ ID.t;

/* Lexemes Same as before and not specified again. */

/* Parsing Declarations */

PROTOTYPE
::= (PSDL_COMPONENTS)

PSDL_COMPONENTS
::= (PROTOTYPE.t = PSDL_COMPONENTS.t)

COMPONENT
::= ()

TYPE_SPEC
::= ()

OPTIONAL_INTERFACE_LIST
::= ()

INTERFACE
::= ()

ATTRIBUTE
::= ()

OPTIONAL_REQUIREMENTS
::= ()

TYPE_DECL
::= ()

OPTIONAL_TYPE_DECLARATION
::= ()

ID_LIST
::= ()

ID
::= ()
IV. SYNTAX-DIRECTED EDITOR (SDE)

As discussed in the previous chapter, the SDE is an editor tailored to PSDL so that it can assist the user in writing correct programs. The power of such a tool is immense and previous work combined with the work of this thesis have only begun to tap its potential. Because the focus of this thesis is on the SDE, it will be covered somewhat extensively here.

This chapter will cover creating an example prototype with the SDE, the constraint checking functionality that is to be added to the SDE, the actual modifications to implement that additional functionality and finally, an example with the modified editor.

A. EXAMPLE OF SDE

1. Simple Tutorial

This section will provide you with the minimal tools necessary to perform software development through rapid prototyping. Your attention will be focused on the PSDL Editor consisting of three separate parts; the Syntax Directed Editor (SDE), the Graph Viewer and the Graphic Editor.

These three parts taken together allow the designer to create the CAPS data flow diagram and PSDL program. They further allow assignment of all timing and control constraints to prototype components which consist of operators and data streams. For a more detailed presentation, see the CAPS Tutorial.

To start CAPS, you simply type "caps". This will start it up in the designer mode. Using the switch "-m" allows CAPS to be started up in the manager mode. We will not use that, as it is beyond the scope of this example. All of the work that is generated under CAPS is saved in a "caps" directory which is directly under your Home directory. If you do not have it, CAPS will create it for you. The current version of CAPS not only creates the "caps" directory, if you don’t have it, but it also copies two sample prototypes into it. One of these is the one you are about to create. Therefore, to avoid this potential problem, if you don’t have a "caps" directory, create your own. The results of this section will produce a design that looks like Figure 20.
Figure 20. Temp_Controller Prototype displayed in Graph Viewer.

As soon as you start CAPS, you will see an interface like that shown in Figure 21.

<table>
<thead>
<tr>
<th>CAPS (designer mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype</td>
</tr>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Computer-Aided Prototyping System

Figure 21. The CAPS User-Interface (Designer Mode).

To select a prototype that already exists, you simply select "Choose" from under the "Prototype" menu, and the available prototypes will be displayed for you to select from. That only selects the prototype. You must then select "PSDL" from under the "Edit" pull down menu to invoke the SDE.

To start a new prototype, pull down the "Prototype" menu from the CAPS designer mode screen and select "New". Do that now and the window displayed in Figure 22 will appear.
Left click the prototype name box which will cause it to become highlighted. Enter TEMP_CONTROLLER and left click your mouse on the OK button. You must use the mouse to activate the name box and acknowledge ok when finished. A <return> will not work here. In addition, CAPS is case sensitive because the underlying file structure is based on UNIX. Identifiers of two words or more must be connected by an underscore. When you do this, two windows will open up as shown in Figure 23.

Notice that the initial root operator in the SDE is the same as the name that you provided in the new prototype selection. TEMP_CONTROLLER is entered as a Stub in the SDE and acts as the operator for the total prototype. Also, the Graph Viewer is empty as we have not started to graph the prototype yet. That's next.
All of the pull-down menus in the SDE are active, however all of the necessary commands for PSDL editing and file saving are found in the “CAPS-Cmds” pull down menu. Do not use the “file” pull-down menus, because if you do SDE will not save a correct PSDL program.

From the SDE, invoke the CAPS Graphic Editor by using the “edit graph” command from the “CAPS-Cmds” pull-down menu. Figure 24 will appear.

![Graph Editor Diagram]

**Figure 24. The CAPS Graphic Editor.**

Once in the Graphic Editor, the designer draws the data flow diagram, enters operator names, inserts input and output streams, and enters some of the prototype timing information. Additional timing and control constraints are entered and implementation options are selected in the SDE. That will be covered after you complete the graph of the prototype.

The CAPS Graphic Editor is used first and foremost to lay out the data flow design of a prototype. Operators are linked together with data streams and given names. Context sensitive attributes are assigned to operators and data streams. These attributes are the maximum execution
time (MET) for operators and latency (LAT) for data streams. Recal that the MET is the maximum amount of CPU time the operator can use for execution and the latency of a data stream is a lower bound on the amount of time required for transmission of data along that stream. The figures and words that appear on the left hand side of the Graphic Editor (the Graphic Editor palette) are editing tools.

The functionality of the tools are summarized as follows:

CIRCLE.....Draw circular operators to represent proposed software components,
SQUARE...Draw rectangular operators to represent simulations of external systems,
LINE........Draw data streams,
Properties...Assign properties to the selected operator or data stream,
Select........Enable selection of an object in the graph.

The name of the active tool is displayed in the lower left portion of the Graphic Editor and the name of the operator being edited is displayed in the lower right portion.

Begin by putting in the software operators for the TEMP_CONTROLLER prototype. Select the Circle Tool, or software operator palette, and left click your mouse. Now move your mouse cursor into the working area and left click again. A circle will appear. Make six more by simply left clicking in six separate places so that your graph looks something like that of Figure 25.
Whoops, you should only have four circles in your graph. You can delete some of them with the select tool! With your mouse, left click on the Select tool. Now move the mouse to a circle and left click again. The circle that you selected will be framed by square reference points. To delete the circle, press the delete key or backspace key on your keyboard. To select another object, simply move the mouse onto another circle and left click again.

You can also move and resize objects within your working space with the use of the select tool. Simply left click on the Select tool and then left click on the object you want to move or modify. Once you have left clicked on the select tool you can also make another object active by clicking on it without hitting the select button again. By depressing the left mouse button while on an object, and then moving the mouse, you can drag the object to where you want to place it. When you reach the point where you wish to place it, simply stop dragging the mouse and release the left mouse button. In the case of re-sizing, simply left click one of the reference points after selecting the object and move the mouse, letting go of the bottom when the object is as desired.

To label your software operators, click on the Select tool, select an object, and then click on the Properties tool. This will bring up the Properties_popup window seen in Figure 26.
Click in the Operator Name box and type its name. Then click in the next box or hit the tab key to enter the MET information, if desired. Now click “OK” or hit the <enter> or <Return> key. The units of milliseconds (ms) are automatically appended to the number entered in the Properties popup dialog box. If you enter units while in the Properties popup dialog box, your max execution times will not appear on the graph.

The TEMP_CONTROLLER prototype with all software operators labeled should appear similar to Figure 27 below.
Max execution time labels can be selected and moved like all other units. Be careful that you do not misplace them however, as the object they refer to should be obvious by their placement.

Next you will use the Line tool to enter data streams between objects. Move your mouse to the Line tool palette and left click. To start the stream, place the cursor in an object and left click. To end the stream, move the mouse to another object and left click again. Data Streams can originate or terminate in an object or outside of an object if they are external input or output. To start or end a stream outside of an object, double click the left mouse button. If curved lines are needed, create way points along the line by clicking the left mouse as you proceed to the termination point of the stream. When the line is completed it will curve to conform with the way points. Figure 28 shows the TEMP_CONTROLER prototype with data streams entered.

![Graph Editor](image)

**Figure 28.** TEMP_CONTROLER with Data Streams entered.

The data streams have properties similar to the other objects in that they have object names and a latency attribute. The Properties_popup for data streams is shown in Figure 29. Again, the latency attribute is measured in ms by default. The functional restrictions pertaining to
data entry for this window are the same as in the Properties popup for operators. Note that the editor is smart enough to call up the right Properties_popup box.

![Properties popup](image)

**Figure 29. Properties Box for Data Streams.**

The data stream popup box allows for indicating whether the data stream is a state stream or non-state stream. These are optional and have not been entered here. In the graphics editor, the default is non-state stream. In this edition of CAPS Release 1, the state streams are defined in the SDE. For more details on data streams, refer back to chapter two.

Data stream labels can be moved like other objects by using the Select tool and then dragging with the mouse. If a data stream name or latency label is deleted the entire data stream is deleted as well. If a stream must be modified (i.e. rename it or change its latency time), it should be done through the Properties_popup box.

The prototype data flow graph is complete and there is little left to do in the graphic model. We will now open up the SDE. To return to the SDE from the Graph Editor, drop the “Graph” menu on the menu bar and select “Return to SDE”. The Graphic representation of your prototype will be saved automatically. With the possible exception of “Decompose”, which will be discussed later, the other choices on both menus are self-explanatory.

The first thing you should notice when you return to the PSDL Editor (See Figure 30) is that stubs have been created (in alphabetical order) for all of the data streams that you created in
the Graph Editor. Stubs, in the form of Control Constraints, have also been entered for all Operators that you created. These too are in alphabetical order.

Figure 30. PSDL Editor after Graph is complete.
Once in the editor, you can use the mouse to move the cursor, which appears as a caret, or you can move it with the arrows on your keyboard. When you move the cursor to the end of "SPECIFICATION", the menu in Figure 31 appears at the bottom of the editor. Some of these selections are input via the Graph Editor, i.e., o_inputs_list, o_outputs_list and o_timing_info. You can enter these things in the SPECIFICATION, but if the information that you enter is inconsistent with that entered in the Graph Editor, the SDE will change it to make it consistent. Most important to remember is that the cursor is context sensitive and menus will automatically open based on where the cursor is placed. You will also know what part of the prototype is currently selected by the underlining that SDE inserts.

![Figure 31. Specifications Tool Bar.](image)

Since we are in the top level of the editor, the only things that we will enter into the SPECIFICATION are key words. Select "o_keywords", which stands for optional keywords; the "o" in all of these options stand for optional. The next display should look like that of Figure 32.

![Figure 32. Keyword Tool Bar.](image)

Note that the context of the cursor position now changes from operator_spec to o_keywords. When the Keywords Tool Bar opens, the SPECIFICATION section of your PSDL editor will change. A place holder will appear in the form of square [] or angel brackets <>. Any time you see square brackets, information placed inside of them is optional. The actual insertion under the SPECIFICATION line should be "[optional keywords]". Click on the Keywords option in the Keyword Tool Bar and the [] will be replaced by <identifier>. You are now ready to enter in

53
Keywords. Hit the return key after each keyword (commas will be inserted). Multi-word keywords can be separated by underscores. When you are finished you must hit return twice. This will save your entries.

If you make mistake, you can back space over incorrect text just entered or you can highlight (click of left mouse) that section of the prototype to be deleted (recall it will be underscored). Then hit <ctrl> <shift> <k> simultaneously. If you have an optional empty place holder you can eliminate it by moving the cursor up with the arrow key or by clicking elsewhere with the left mouse button.

Other items that you can add to the SPECIFICATION are informal descriptions and STATE DECLARATIONS. The informal descriptions will appear in { } and are similar to user/programmer comments in other programming language. They are for human consumption only and are ignored by the PSDL compiler. The State Declarations are necessary to specify Data Streams that require initial values.

You can look at the graph at anytime by clicking in the PSDL where there is mention of the graph components. For example, positioning the cursor at the "OPERTOR TEMP_CONTROLLER" or below will bring up the graph at the top level (the only level at the current time). The Graph Viewer will come up automatically. If the graph viewer window is closed (or minimized), it will be labeled, "graph_edit". Do not be confused by this as it is still the graph viewer.

Now enter the declaration types for the data streams. Position the cursor before, in, or after the <decl_type_name> component of the DATA STREAM. A subtle requirement of the SDE should be mentioned here. When entering information applying to streams and control constraints of operators, you must enter them into the parent of the operator or stream. Left Click the place holder specified above and a menu bar of standard defined types will activate at the bottom of the PSDL Editor. Click on the type declaration "FLOAT" and that data stream will be defined for that type. Now do the other two. Actually, you would pick which ever type makes the most sense for your design. User defined types can also be accessed from this menu bar and typed in by the user but that is outside the scope of this guide.

More important however is the propagation of type declarations throughout the PSDL program when you define the data streams. Once you’ve selected the types for these three streams
within the parent operator, you merely left click anywhere else and the new declarations are propagated throughout the prototype as shown in Figure 33 below.

Figure 33. Complete PSDL Program showing Type Declaration propagation.

Also illustrated in Figure 33 is the PSDL Editor’s ability to correctly assign data streams as either input or output within the Operator Specifications found in the top of PSDL program. The Maximum Execution Times that you entered in the Graphics Editor are displayed there as well.

Control Constraints can be modified directly from the PSDL Editor by using the mouse or arrow keys as previously mentioned. While in the graphics editor, two processes (Sensor and Evaluate_Temp) were deemed as time-critical and were given MET’s. At this point, it must be decided whether they are periodic or sporadic operators. Suppose we decide that both should be periodic and that the constraints are as follows: \( \text{PER}_{\text{Sensor}} = 190\text{ms} \), \( \text{PER}_{\text{Evaluate}_\text{Temp}} = 210\text{ms} \), \( \text{FW}_{\text{Sensor}} = 185\text{ms} \), and finally \( \text{FW}_{\text{Evaluate}_\text{Temp}} = 205\text{ms} \).
In order to accomplish this, we first left click just past the control constraint we wish to edit; “Sensor”. Then the menu bar at the bottom of SDE changes to that shown in Figure 34. You click on “optional_period”, and then the menu in Figure 35 appears. Click “Optional_Period” and the operator “Sensor” has its first constraint under it. You do the same procedure for entering the time (not shown here); i.e. left click “Time_Expression”, enter an integer for the placeholder (should be 190), hit return, select the desired units from the menu bar, and finally click where you want to go next.

![Figure 34. Control Constraints Menu Bar.](image)

![Figure 35. Optional Period Menu Bar.](image)

After you enter in the constraints for the two time-critical operators, your PSDL should look like that pictured in Figure 36.
Figure 36. Complete PSDL Program with Control Constraints.

But what if we wanted to decompose one of our operators? Recall the CAPS prototyping process from chapter two. When you have a component, you first look to see if it can be implemented with a component from the software base. If not, it must be determined whether the operator should be decomposed. If yes, you will go through the same thought process just described to specify the sub-components of the operator. When dealing with operators that should not be decomposed, they must be implemented by hand (currently requires Ada implementation). So then, the question here is how to decompose an operator.

First select “edit-graph” from the “CAPS-Cmds” in the SDE menu bar. You should get a window that looks like Figure 28 again.
Figure 28. (Repeated).

Select the Sensor Operator and then select "Decompose" from the "Graph" menu bar. It will automatically open up a Graph Editor that looks like Figure 37 below. The Graph Editor shows the data stream, "Temperature", as an external output from the Sensor operator. This is information that you need to keep in mind as you decompose the Sensor. It must be maintained for consistency and is brought forward so that the user does not have to look back at the parent operator.
Figure 37. Operator Sensor after initially selecting decompose.

To decompose Sensor, you draw the appropriate data flow diagram just as before. This diagram, however, is the internal workings of the composite operator "Sensor". An example is given below in Figure 38.
Figure 38. Operator Sensor decomposed.

Now that you are done with the editor, you can either go back to the previous level data-flow graph or return directly to SDE (choose to go back up this time). Those options are both under the "Graph" pull-down menu as "Edit Parent" and "Return to SDE" respectively. If you go back to the top level diagram, you should notice that the "Sensor" operator is different as shown in Figure 39. This doubled circle denotes that the operator is composite and has been decomposed.
Figure 39. TEMP_CONTROLLER Prototype with Sensor marked as decomposed.

The PSDL file in the SDE will reflect the changes made and will add new operators, external simulators and data streams as necessary. The declarations and control constraints for these objects are modified as previously described in this guide.

You should now be able to build a prototype in the graphic editor, make type declarations, and define control constraints in the SDE. The following table provides a quick reference on which type of information can be entered into each editor.

<table>
<thead>
<tr>
<th>Graphic Editor Information</th>
<th>Syntax Directed Editor Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertices &amp; edges (operators &amp; data streams)</td>
<td>control constraints</td>
</tr>
<tr>
<td>operator names</td>
<td>data stream types</td>
</tr>
<tr>
<td>data stream names</td>
<td>timer declarations</td>
</tr>
<tr>
<td>operator maximum execution time</td>
<td>state declaration &amp; initialization</td>
</tr>
<tr>
<td>data stream latency time</td>
<td>user defined types</td>
</tr>
<tr>
<td>operator color &amp; shape</td>
<td>operator &amp; type implementation selection</td>
</tr>
</tbody>
</table>

Table 2. Summary of editor inputs.

If you look back at the complete PSDL program in Figure 36, you will notice one more set
of placeholders that haven’t been filled in. These are for the “operator implementation”. As discussed earlier, for each component in your design, you must either find a component within the software base that can fill its needs, further decompose it, or implement it in Ada. For instructional purposes here, simply go to each placeholder, click on it, and select “Ada_Implementation”. Obviously, if this were a real application, someone would eventually have to implement those atomic operators. Once the prototype is complete, save your work in the PSDL Editor by “PSDL-Save” or “PSDL-Save-Exit” from the “CAPS-Cmds” button on the menu bar. The next step would be to Translate the prototype to generate the code that will tie everything together. After that, you would invoke the Scheduler which determines a schedule for the system. And finally, it would be compiled to create executable code. Don’t go any farther as there are problems, besides the fact that you don’t have implementations for your operators, that will be covered in section C.

2. SDE Menu Functions

Because the last section was designed to be a simple example, the menu of the SDE was mostly excluded with the exception of some of the options under “CAPS-Cmds”. Recall that the “File” menu should not be used. This section was added because there are several options that the designer should be familiar with. The others can be experimented with to get some exposure as free time dictates.

Under the “Edit” pulldown menu are several options that most users are already accustomed to. The cut, copy, paste, and delete are in most editors today. The SDE is no exception. This menu gives you a choice of either text or a structure that you’ve already highlighted, and acts like any other editor with respect to use. Getting accustomed to these can enhance the designer’s productivity as they speed development time.

The “View” pulldown menu deals with what representation of the abstract syntax tree you wish to see. You can select any number of different views or a combination of them. While this is fun to experiment with, the default view is the best for the beginning designer.

The “Tools” pulldown menu is also for a more advanced user. A designer that is familiar with the concepts of the sub-menu items below this option will quickly grasp their intent as they are very common in the Unix toolset.

The “Structure” pulldown menu deals with moving around in the edited prototype. Most of this is also accomplished more easily with the tab, space bar, and especially the mouse. The
one option that must be mentioned, however, is the "ascend-to-parent". Sometimes the user can get confused as to what part of the abstract syntax tree is being displayed in the editor and there are times when the structure highlighted must be walked back up the tree. For example, when a designer decides to quit work on a prototype before assigning types to the streams. Before quitting, the prototype would look something like that displayed in Figure 40 below. Notice that the placeholder says `<decl_type_name>`.

![Figure 40. Data Stream without Type Declaration.](image)

If the designer were to exit the SDE, saving the prototype without fully defining the stream S1, the placeholder would be different upon re-editing the prototype. Because the SDE checks to ensure that any prototype being loaded into the editor is syntactically correct, the stream must be defined. To keep from getting an error then, SDE puts an identifier in the placeholder. Now when you open the prototype, you see what is displayed in Figure 41.
The designer may now want to make the stream an integer. Instead of indicating a type_decl_name as before, it shows the placeholder as an id. The designer must select the id, UNDEFINED_TYPE_NAME, and delete it using the “delete-structure” option in the “Edit” menu. This will make the placeholder appear as <identifier>. The next step is to walk up the syntax tree. This is done by selecting “ascend-to-parent” in the “Structure” menu. After doing that, you should notice that the context (shown in the Help Pane at the bottom of the SDE) now shows the placeholder to be a decl_type_name, which is desired. Delete the placeholder one more time and you are back to what was there before you quit. In addition, the Help Pane now shows all the available transformations for a decl_type_name.

Alternately, one can first select the id, UNDEFINED_TYPE_NAME, and then “ascend-to-parent” in the structure menu, followed by selecting the “delete-structure” option under “Edit”.

The “Text” pulldown menu provides more ways of moving around, provides for search
and replace techniques, and has an undo option. And finally the help is the last pulldown menu but is very limited in its current implementation.

Finally, some of the useful keystrokes of the editor are listed below. Recall that the delete keystroke was covered earlier in the example.

Control-Shift-K......................Delete Structure,
Control-Shift-P.........................Move up one level in the parse tree,
Control-h..............................Delete previous character,
Control-v..............................Advance one page,
Control-d..............................Delete next character,
arrow keys..............................move up, down, left, right,
delete or backspace....................Delete previous character.

There is no “undo” command in the Syntax Directed Editor, so the user must be careful. If a major problem occurs during editing, the best option is to select “exit” from the “CAPS-Cmds” pulldown menu and exit without saving.

B. NEW FUNCTIONALITY

1. Scheduling Constraints

   a. General

   All time-critical operators are non-preemptable under the current CAPS model, meaning that once they start, they will run until done. And as previously mentioned, these operators are statically scheduled; before runtime. The other operators, however, are preemptable, and are scheduled dynamically at runtime. This means the hardest part of ensuring that a real-time system will run correctly is ensuring those time-critical operators can be scheduled. Non time-critical operators simply use whatever time is left during runtime, including the time left over when an operator finishes early. For that reason, most of the concentration here will be on time-critical operators.

   Recall that when an operator has a MET (maximum execution time), it is time-critical and is classified as periodic or sporadic. The previous definition of MET, the maximum amount of CPU time the operator can use for execution, is right on for atomic operators. In the case of composite operators, however, MET is defined a little more loosely as the maximum time needed along any thread of control within the operator itself. In addition to the above definition, it must be realized that the MET encompasses data triggering checks, stream reads, execution guard checks, execution of the operator, output guard checks, stream writes, and exception handling [Ref. 11]. Table 3 recaps the two classifications of a time-critical operator.
Table 3. Classifications of Real-Time Operators.

As shown above, and previously discussed in chapter two, periodic operators also have a PER (period), and a FW (finish within) time where the PER depicts the frequency in which the Scheduler makes a processor available for execution and the FW denotes how long from the start of the PER before execution must be completed. Figure 42 below is provided to help make this concept a little clearer. The figure shows that periodic operators are triggered by temporal events occurring at regular intervals; thus the PERiod, which is the span of time between two successive activations.

Figure 42. Timing Constraints for the Periodic Operator.

In addition to the MET, a time constrained operator that is sporadic has a MRT (maximum response time) and a MCP (minimum calling period), where MRT refers to the maximum amount of time between the arrival of new data on the input data stream(s) (which triggers the operator) and the time when the last output is put on the output data streams, and
MCP refers to the lower bound on the delay between two subsequent arrivals of triggering data on the input. MCP is actually more of a constraint on the operator(s) that produce a triggering data stream. Figure 43 shows sporadic operator constraints pictorially.

**Figure 43. Timing Constraints for the Sporadic Operator.**

There are two other timing constraints provided by PSDL that haven't been mentioned yet. They are for dealing with distributed systems, and because they are important, are covered here. However, these are not used in the current enhancements to the SDE and therefore will not be mentioned after this paragraph. The first is *Latency* (LAT), which denotes the maximum delay that can occur between a producer operator writing to a stream and a consumer operator reading that same stream. This can become a problem when dealing with networks and therefore must be accounted for in such cases. The second constraint for distributed systems is the *Minimum Output Period* (MOP), which is the amount of time an operator must wait before issuing another write to an output stream. Both of these constraints are due to the possible delays introduced by network traffic, and restrict an operator from reading/writing from/to a stream before the appropriate LAT/MOP has elapsed.

b. Data Triggering Semantic Checking

The specific constraints that must hold in order that a prototype is schedulable are many and complex. Because of this, only the simpler ones that are directly involved in changes
to SDE are mentioned. Also, because the focus of this thesis is concerned with making SDE more productive by adding schedulability checking, the theories behind schedulability rules will be discussed lightly and not proved. For a much more in-depth look at the fundamental theory of the scheduling of distributed hard real-time systems, see reference [Ref. 11, 16].

After gaining some familiarity with the concept of periodic and sporadic operators, several relatively simple, but important, constraints that must be imposed on the producer and consumer operator connected via one or more streams that need to be adhered to become apparent; constraints that can be checked while in the SDE.

If the producer of a stream has a period that is shorter than that of the consumer of the same stream, there is a strong likelihood that a problem will develop whereby the consumer can not stay caught up. For that reason, the rule is that the Consumer’s PER must be less than or equal to that of the Producer if they are connected by a dataflow stream.

Sporadic operators are not regularly firing operators. They fire based on the arrival of new data. However, if a sporadic operator has no data trigger, it will be defaulted to a periodic operator with period equal to MCP to meet the need for some sort of conversion for periodicity and thus schedulability.

The next constraint involves a time-critical producer with a non time-critical consumer connected by a dataflow stream. There is a strong likelihood that the high priority producer will overflow the dataflow stream because the consumer can fire only when free time is available. Having the producer put data on the dataflow stream regularly with the consumer using the data when time is available simply will not work.

The last two constraints to be checked by the SDE involve the situation where a stream is produced by a non time-critical operator and consumed by a time-critical operator. If the consumer is triggered with BY ALL, the designer should be warned that possible overflow could result because if the producer gets enough time, it could produce new data for the stream before the next execution of the consumer has commenced. Finally, if the consumer is triggered with BY SOME, the designer should be warned that possible data loss could result because, just as before, the producer could get enough free time that it writes to the stream before the consumer reads the old data; the difference being that this time the old data would just go away.

Table 4 below shows the semantic data trigger related checking that is accomplished by the SDE. As alluded to earlier, cases not discussed nor shown below are either
trivial ones that call for no conditional checking or those which require sporadic operators to be converted. As SDE does not do any converting of sporadic operators for semantic checking, these can not be checked. Those cases can be found, however, in reference [Ref. 11].

![Diagram](image)

\[ P = \text{Periodic Operator} \\
S = \text{Sporadic Operator} \\
NTC = \text{Non Time-Critical} \]

<table>
<thead>
<tr>
<th>Op1</th>
<th>Op2</th>
<th>Data Trigger</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>P</td>
<td>By All</td>
<td>If ( \text{PER}<em>{\text{Op1}} &lt; \text{PER}</em>{\text{Op2}} ) then Error: Producer PER can't be less than that of Consumer</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td>By Some</td>
<td>If ( \text{PER}<em>{\text{Op1}} &lt; \text{PER}</em>{\text{Op2}} ) then Warning: Possible Data Loss</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td>None</td>
<td>If ( \text{PER}<em>{\text{Op1}} &lt; \text{PER}</em>{\text{Op2}} ) then Warning: Possible Data Loss</td>
</tr>
<tr>
<td>P</td>
<td>S</td>
<td>None</td>
<td>Warning: Sporadic Consumer Operator will have MCP as default period</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>None</td>
<td>Warning: Sporadic Operators will have MCP as default periods</td>
</tr>
<tr>
<td>P</td>
<td>NTC</td>
<td>By All</td>
<td>Error: Possible Stream Overflow</td>
</tr>
<tr>
<td>P</td>
<td>NTC</td>
<td>By Some</td>
<td>Warning: Possible Data Loss</td>
</tr>
<tr>
<td>S</td>
<td>NTC</td>
<td>By All</td>
<td>Error: Possible Stream Overflow</td>
</tr>
<tr>
<td>S</td>
<td>NTC</td>
<td>By Some</td>
<td>Warning: Possible Data Loss</td>
</tr>
<tr>
<td>NTC</td>
<td>P</td>
<td>By All</td>
<td>Error: Possible Stream Overflow</td>
</tr>
<tr>
<td>NTC</td>
<td>P</td>
<td>By Some</td>
<td>Warning: Possible Data Loss</td>
</tr>
<tr>
<td>NTC</td>
<td>S</td>
<td>By All</td>
<td>Error: Possible Stream Overflow</td>
</tr>
<tr>
<td>NTC</td>
<td>S</td>
<td>By Some</td>
<td>Warning: Possible Data Loss</td>
</tr>
<tr>
<td>NTC</td>
<td>S</td>
<td>None</td>
<td>Warning: Sporadic Consumer Operator will have MCP as default period</td>
</tr>
</tbody>
</table>

Table 4. Semantic Data Triggering Constraints to be done by SDE.

c. Timing Constraint Semantic Checking

The timing constraints attached to a time-critical operator are subject to multiple restrictions to ensure the possibility that the prototype is schedulable. As already mentioned, there are more than is described here; cases which fall in either the trivial or too hard category. Below is the first of several restriction that must hold.

\[ \text{MET} \leq \text{FW} \leq \text{PER} \]
If the MET was allowed to exceed the FW, the operator could use more CPU time than its deadline allows; this clearly can not be allowed. The second inequality must hold for a single CPU schedule because if the operator was allowed to finish after the period was over, it would delay the start of that operator’s next activation.

The Maximum Execution Time Theorem [Ref. 11] states the following must hold for the set of all periodic operators in order that the prototype be schedulable on a uniprocessor.

\[ \text{MAX (MET)} \leq \text{MIN(PER)} \]

This in no way guarantees that the prototype is schedulable; we only know that it isn’t if this is violated. There are two possibilities here.

In the case where the same operator had constraints such that MET > PER, it would not be schedulable even with a multiprocessor; that case is covered above.

In the second case, where the MET and PER belong to different operators, the constraint becomes apparent when you think about the fact that when violated and dealing with only one processor, the operator with the smaller PER will be blocked longer than the span of its PER while the operator (remember these are uninterruptable) with the larger MET executes. If the second operator is blocked longer than its PER, it can’t possibly make the deadline. So then it makes sense that the smallest PER must be greater than the largest MET.

The first question that comes to mind is, “How many processors will do the job?”. This question is answered for periodic operators with the Load Factor, which is cited in many sources dealing with scheduling. It shows the minimum number of processors that will be required to schedule the prototype.

\[ \left\lceil \sum_x \left( \frac{\text{MET}_x}{\text{PER}_x} \right) \right\rceil \]

Again, this does not guarantee schedulability. It only guarantees that without the number of processors specified, the prototype is not schedulable. Intuitively, this formula states that the larger the MET is with respect to its PER, the more utilization is required of the processor. This means, as they get closer to being equal, the operator gets closer to needing a processor all to itself.

Static scheduling requires having advance knowledge of the process behavior. This means all operators must be effectively periodic [Ref. 12], and further means sporadic
operators must be converted so that they have equivalent periods. The process of attaining the equivalent period is somewhat complex, and involves heuristics. The outcome is never greater than min((MRT - MET), MCP), and so this can be used as an upper bound. Using this information and other theorems presented in reference [Ref. 11], the following constraint is required of all sporadic operators.

\[ 2 \times MET \leq MRT, \quad MET \leq MCP \]

Presented thus far are the constraints that are imposed on otherwise valid time-critical operators. There are other timing constraints that are somewhat more intuitive.

First, every time critical operator must have an MET so that the scheduler can know how much time must be allocated to the operator in the schedule. Therefore, if is an error for an operator to have a PER, FW, MCP, or MRT without a MET.

Second, a time-critical operator must be periodic exclusive-or sporadic. This means it must be one or the other, but not both. Because of this, mixing constraints from both and having both is definitely an error.

Table 5 below shows the semantic checking that can be accomplished by the SDE. Again, cases not shown in the table are either trivial, calling for no conditional checking, or require sporadic operators to be converted. As SDE does not currently do conversion of sporadic operators for semantic checking, these can not be checked. They can be found, however, in reference [Ref. 11].
<table>
<thead>
<tr>
<th>MET</th>
<th>PER</th>
<th>FW</th>
<th>MRT</th>
<th>MCP</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>OK</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Error: Time-Critical Operators must have a MET</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Error: Time-Critical Operators must have a MET</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Error: Time-Critical Operators must have a MET</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Error: Time-Critical Operators must have a MET</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Error: Time-Critical Operators must have a MET</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Error: Time-Critical Operators must have a MET</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Error: Time-Critical Operators must have a MET</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Error: Time-Critical Operators must have a MET</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Error: Time-Critical Operators must have a MET</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Error: Time-Critical Operators must have a MET</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Error: Time-Critical Operators must have a MET</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Warning: MRT and MCP will take default values</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Warning: MRT will default to MCP + MET</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Warning: MCP will default to MRT - MET</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>If NOT(2*MET≤MRT, MET≤MCP) Then Error: Prototype will not schedule</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Warning: PER will default to FW</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Error: Operators can not be both Periodic and Sporadic</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Error: Operators can not be both Periodic and Sporadic</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Error: Operators can not be both Periodic and Sporadic</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Warning: FW will default to PER</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Error: Operators can not be both Periodic and Sporadic</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Error: Operators can not be both Periodic and Sporadic</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Error: Operators can not be both Periodic and Sporadic</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>If NOT(MET ≤ FW ≤ PER) Then Error: Prototype will not schedule</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Error: Operators can not be both Periodic and Sporadic</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Error: Operators can not be both Periodic and Sporadic</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Error: Operators can not be both Periodic and Sporadic</td>
</tr>
</tbody>
</table>

Table 5. Semantic Timing Constraint to be done by SDE.
C. MODIFICATIONS

Modifications to the SDE are based on the additional functionality outlined in the last section. That functionality was summarized in Tables 4 and 5.

1. Data Triggering Constraints

When looking at the remarks of the Table 4, it is apparent that the data triggering constraints can be further simplified into five major points:

1) When the producer and consumer operator are both periodic and connected by a dataflow stream, the PER of the producer must greater than or equal to that of the consumer,

2) When the producer of a stream is time-critical and the consumer is not, if the consumer has a BY ALL trigger, overflow can result,

3) When the producer of a stream is time-critical and the consumer is not, if the consumer does not have a BY SOME trigger, data loss can result.

There is some commonality about these rules that will be exploited in order to minimize the task of modifying the SDE. First, each rule identifies a consumer or producer (or both) of a stream. Therefore, a structure will be needed to store whether an operator is a producer or a consumer. It will also need to keep track of whether the consumer operators have a trigger and what type it is. The rules also depend on the knowledge of whether or not the operator is time-critical. This means a set of time-critical operators must be created so they can be kept track of separately. Further, those time-critical operators must be broken down into a mutually exclusive set of periodic and sporadic operators. And one more piece of information about those operators that is needed is the value of its PER. The astute reader will note that sporadic operators do not have a PER. This means that the sporadic operator should be converted so that it will have an equivalent PER.

Timing Constraints

An inspection of the remarks of Table 5 will also bring to light some commonality that can be summarized. Those rules can be broken down into six major points:

1) A time-critical operator must have a MET,

2) A time-critical operator can not mix periodic and sporadic timing constraints,

3) A time-critical operator with only one of its two timing constraints (FW and PER for Periodic and MRT and MCP for Sporadic) will have the unspecified
constraint set to a trivial, default value,

4) An otherwise valid periodic operator must follow the specification of
   MET ≤ FW ≤ PER or the prototype can not be scheduled,

5) An otherwise valid sporadic operator must follow the specification of
   2*MET ≤ MRT and MCP ≤ MCP or the prototype can not be scheduled.

Just as with the data triggering rules, these must be able to identify whether an operator is
constrained. In addition, they must be able to determine the existence and value of any FW,
PER, MRT, and MCP constraints. The next step is to define structures to hold the information
deemed required above.

2. Abstract Syntax Rule

Summarizing the requirements of both the data triggering rules and the timing constraint
rules three structures will be required. The name of each structure is in bold parenthesis. An
edge set (edge_set) is needed to keep track of all edges. It will store the edge’s name, producer
operator, consumer operator, and latency value. The latency is not needed for these rules, but it
makes sense to keep it, as it could be used in future enhancements to the SDE. The second
structure needed to support these rules is a constrained operator set (op_id_met_set). It will
store the name of all operators that have a MET and the value of the MET, thereby identifying
constrained operators. Finally, a structure will be needed to store all of the constrained operators
and their constraints in a set (id_constraint_set). It will store the name of all constrained
operators, their time constraints (per and fw for periodic operators, and mcp and mrt for sporadic
operators), and type of trigger.

The last two structures, which will be referred to as phyla from here on, look as though
they should be combined. The first attempt at defining additional abstract syntax rules did in fact
have these two together. The problem is that the MET exists in a separate section of the
derivation tree than do the other timing constraints. Therefore, these two distinct phyla need to
exist at the lower levels of the tree anyway. The three additional phyla required are displayed in
Figure 44 below. The phyla not terminating here are further defined in the full abstract syntax
rules of PSDL in Appendix B as a phylum or lexeme. REAL is a built in type.
Figure 44. Additional Abstract Syntax Rules for SDE.

As tree representations are sometimes more familiar and easier to understand, these rules have been re-displayed in the trees of Figure 45 below. While it doesn’t contain as much information as the actual specifications, it may make quickly referencing the rules easier.
Figure 45. Addition Abstract Syntax Rules for SDE (2).
3. Attribute Rules

All of the timing constraints ended up being represented as real numbers as opposed to the normal representation of integers with an unit (i.e. 90 milliseconds). These were represented as real so that no matter what number is given in the design, it could be converted to a default internal representation for comparison purposes. This means that when the auxiliary functions were defined (Appendix E), there had to be one that converts the timing constraint representation within the old abstract syntax to real for compliance with this abstract syntax.

With one exception, the attribute names are simply the same as the newly created phylum with the prefix syn or inh, depending on whether it is synthesized or inherited. The exception is the syn_vertex_id_met_set of type op_id_met_set. The word “vertex” was substituted in for “op” because, while they are the same thing, the previously defined abstract grammar used the term vertex. Therefore, the declared attribute name was changed.

Earlier, it was stated that the MET and other timing constraints were established in different parts of the derivation tree. Figure 46 is a partial look at the abstract syntax rules for the PSDL SDE in tree form. On the tree the attributes equations required for the semantic checking outlined above are shown in bold type. Note where the MET information is located as opposed to where the timing constraint information is (typed within parenthesis at the bottom of the figure). The MET is defined under a_vertex, the edge constraints under an_edge, and the timing constraints under a_constraint. Therefore, that is where the synthesized attributes had to start. While the synthesized attributes bubble up the tree, the inherited attributes sink down the tree. This is evident when looking at the Figure 46. The vertex_id_met_set bubbled up starting at the vert_list phylum, continuing on to the graph phylum and then on to the operator_impl phylum where it is copied to the inh_vertex_id_met_set attribute at the operator_impl phylum. From there, it begins sinking on the right side of the tree; first to the cc phylum and then to the constraints phylum where it can be used for semantic checking.
Figure 46. Abstract Syntax with Attributes Equations.
While it seems a little backwards to show the attribute equations before the attribute declarations, it was done to not only demonstrate what the attribute equations are but more importantly show graphically where they must exist.

Recall that all synthesized attributes are built at the lower levels and then passed up the tree. Also, realize that in lists, every set of attributes, which are defined by the equations in the figure, will exist once for each occurrence in that list. Psdl_components, for example, is composed of a component and another psdl_components (psdl_components$2). The equations under the phylum component (shown in the figure) will synthesize all of its attributes for the mentioned component (shown in the figure) and again for every other component in psdl_components$2, recursively. Therefore, at the component phylum, the set of id_constraints, for example, will be only the ones within that component (its children). At the prototype phylum, however, there will be one set and it will contain all of the id_constraints within the prototype.

The edge_set didn’t need to be collected into one single set for all of them, so it was not synthesized up to the prototype phylum. It stopped at the operator_impl phylum, which then had all of the edges for its child operators. In fact, the semantic checking requires all of the attributes to be collected at the operator_impl phylum. After all, the best place to check and subsequently notify the designer of a problem is at the component where the problem exists. The reason that the other two attributes (id_net_set and id_constraint_set) are synthesized all the way up to the prototype phylum is so that the number of processors required could be determined. It was decided that the Load Factor equation would be used to determine how many processors are required by the design. This, of course, necessitates having all of the MET’s and PER’s in one place. What this also calls for, is being able to convert the sporadic operators. Because there was not enough time to do that function, the number of processors required is only based on periodic operators. The display on the editor must state this disclaimer.

Figure 47 below shows both the attribute declarations and equations for the synthesized and inherited attributes. The local attributes and equations for semantic checking will be discussed next along with the required additions to the attribute rules.
Figure 47. Synthesized and Inherited Attribute Equations of Attribute Rules.

Because the amount of code for implementing the local attributes, equations and auxiliary functions for the semantic checking required is large, only part of the changes to the
SDE will be shown here. Appendix C contains a full listing of the PSDL SDE Attribute Rules, including those added by this thesis. The additions to the operator_impl phylum are significant enough to show some of the work done. It is represented in Figure 48.

```
operator_impl
  | CommonImpl, AdaImpl()
  | OperatorImpl
  |
(local BOOL producerop.period_le_consumerop.error; producerop.period_le_consumerop.error =
  (is_ProducerOp_Period_LE_ConsumerOp($$sync_edge_set,
     $$sync_id_constraint_set));
local STR producerop.period_le_consumerop.msg;
producerop.period_le_consumerop.msg = (producerop.period_le_consumerop.error)
  ? "\n" # "For any edge, the Producer's Period" # -- must exceed that of the Consumer." 
  "\n" # "For any edge, the Consumer is in" 
  "\n" # "Sporadic, it must have a Trigger." 
  "\n"

(local BOOL sporadic_consumerop_no_trigger.error;
sporadic_consumerop_no_trigger.error =
  (is_Sporadic_ConsumerOp_No_Trigger($$sync_edge_set,
     $$sync_id_constraint_set));
local STR sporadic_consumerop_no_trigger.msg;
sporadic_consumerop_no_trigger_msg =
  (sporadic_consumerop_no_trigger.error)
  ? "\n" # "For any edge, if the Consumer isn't" 
  "\n" # "Sporadic, it must have a Trigger." 
  "\n"

(local BOOL constr_producerop_and_unconstr_consumerop_w_trigger.error;
constr_producerop_and_unconstr_consumerop_w_trigger.error =
  (is_Constr Producerop_And_Unconstr Consumerop_W_Trigger($$sync_edge_set,
     $$sync_vertex_id_net_set, $$sync_id_constraint_set));
local STR constr_producerop_and_unconstr_consumerop_w_trigger.msg;
constr_producerop_and_unconstr_consumerop_w_trigger_msg =
  (constr_producerop_and_unconstr_consumerop_w_trigger.error)
  ? "\n" # "For any edge, if the Producer is constrained,\n" # "an unconstrained Consumer can not have a Trigger." 
  "\n"

(local BOOL unconstr_producerop_and_constr_consumerop_w_all.error;
unconstr_producerop_and_constr_consumerop_w_all.error =
  (is_Unconstr Producerop_And_Constr Consumerop_W_All($$sync_edge_set,
     $$sync_vertex_id_net_set, $$sync_id_constraint_set));
local STR unconstr_producerop_and_constr_consumerop_w_all.msg;
unconstr_producerop_and_constr_consumerop_w_all_msg =
  (unconstr_producerop_and_constr_consumerop_w_all.error)
  ? "\n" # "For any edge, if the Producer is unconstrained,\n" # "a constrained Consumer triggered By All\n" # "can result in overflow." 
  "\n"

(local BOOL unconstr_producerop_and_constr_consumerop_w_byone.error;
unconstr_producerop_and_constr_consumerop_w_byone.error =
  (is_Unconstr Producerop_And_Constr Consumerop_W_Byone($$sync_edge_set,
     $$sync_vertex_id_net_set, $$sync_id_constraint_set));
local STR unconstr_producerop_and_constr_consumerop_w_byone.msg;
unconstr_producerop_and_constr_consumerop_w_byone_msg =
  (unconstr_producerop_and_constr_consumerop_w_byone.error)
  ? "\n" # "For any edge, if the Producer is unconstrained,\n" # "a constrained Consumer triggered By One\n" # "can result in Data loss." 
  "\n"

(local BOOL has_error;
has_error =
  (producerop.period_le_consumerop.error
    || sporadic_consumerop_no_trigger.error
    || constr_producerop_and_unconstr_consumerop_w_trigger.error
    || unconstr_producerop_and_constr_consumerop_w_all.error
    || unconstr_producerop_and_constr_consumerop_w_byone.error);
local STR error_header;
error_header = (has_error)
  ? "\n" # "------------------------------------------\n" # "SCHEDULING NOTICE:" 
  "\n" local STR error_trailer;
error_trailer = (has_error)
  ? "\n" # "------------------------------------------" 
  ""
```

Figure 48. Local Attributes and Equations for operator_impl Phylum.
The code is fairly straight forward and separated into sections divided by dashed lines. Which section implements which semantic check can be quickly determined by looking at one of the two possible string values that are assigned. Each correspond with the remarks of Table’s 4 & 5, and are mentioned again at the beginning of this section. The last section in Figure 48 merely sets up the header and trailer if any of the previous constraint checking variables are true (indicating violation).

Many functions are referenced in the last two figures. The previously existing auxiliary functions are listed in Appendix D, while the auxiliary functions written in support of this thesis are in Appendix E. Recall that function names, for the conventions here, are mixed upper and lower case with underscores; i.e. This_Is_A Function_Name.

To ensure that the functions are understood, one will be covered here. The first section of code in the last figure checks to ensure that for any edge, if the consumer operator’s period is greater than or equal to the producer’s, an appropriate error message is displayed. The boolean variable is set to true, setting the displayed string to the error message vice null, if the function IsProducerOP_Period_LE_ConsumerOP returns true. This is a boolean function that takes two arguments (operator_impl.syn_edge_set, operator_impl.syn_id_constraint_set), and is displayed in Figure 49.

```c
BOOL exported
IsProducerOP_Period_LE_ConsumerOP(edge_set es, id_constraint_set cs) {
    with(es) {
        EdgeSetNil: false,
        EdgePair(hd, tl): with(hd) {
            EdgeNull: IsProducerOP_Period_LE_ConsumerOP(t1, cs),
            Edge[p, c]: ProducerNull: IsProducerOP_Period_LE_ConsumerOP(t1, cs),
            Producer[p_opid]:
            with(c) {
                ConsumerNull: IsProducerOP_Period_LE_ConsumerOP(t1, cs),
                Consumer[c_opid]:
                Get_Period[p_opid, cs] == PerNull ||
                Get_Period[c_opid, cs] == PerNull ||
                ? IsProducerOP_Period_LE_ConsumerOP(t1, cs)
                : (Get_Period[p_opid, cs] <= Get_Period[c_opid, cs])
                ? true
                : IsProducerOP_Period_LE_ConsumerOP(t1, cs)
            }
        }
    }
    return true;
}
```

**Figure 49. Auxiliary Function.**

As mentioned in chapter three, if not familiar with recursion, this would be a good time to review. Most of the functions take on this same format. The keyword "with" is used like a case statement that covers all of the operators of the with’d phylum, and most of them
recursively search sets such as the edge_set or id_constraint_set. When looking at a function such as this, ensure that the phylum description (abstract syntax) of the attributes passed in are close by. This makes reading the function much easier.

The displayed function first looks at the edge_set. If it is nil (empty), the answer is false (there is no edge in error). If the set is not empty, the first edge or the head (an_edge) is pulled off and its producer and consumer operator_id’s are determined. Then, the function Get_Period is used with those operator_id’s to return the period’s of each and are compared to see if either is null. If that is the case, or if a nullary value is hit prior to this point, the function is simply called again with the same id_constraint_set and the rest of, or tail of the edge_set. If there are two valid periods assigned, they are checked to ensure that the producer’s exceeds the consumers. If not, true is returned (thus an error). If so, once again, the function is called with the tail of the edge_list. This will continue until an error is reached or until the end of the list is reached when EdgeSetNil is found and false is returned.

4. Unparsing Rules

The unparsing rules only called for a few additions. Recall, theses are the specifications that control what is displayed by the editor. The only items that must be displayed and weren’t already are the additional local string attributes that were defined in the attribute rules to either display the error that exists or a null value when no error exists.

Figure 50 shows the unparsing for the operator_impl phylum because that is the one displayed with its attributes in the last figure. Notice that all of the local string attributes declared and built in the attribute rules are displayed under the operator named OperatorImpl. The various views represent different unparsing schemes so that what is seen by the user depends on which view is being displayed and multiple views can be displayed at once if desired. Also notice that comments are placed after each placeholder to help identify which phylum is represented by that placeholder.
5. Transformation Rules

The transformation rules of the SDE were not modified by this thesis and therefore are not discussed here. Appendix F has the transformation rules for the PSDL SDE.

6. Concrete Rules

The concreted rules were not modified either and therefore are not discussed but are fully listed in Appendix G.

### D. TESTING

In the beginning of this chapter, a prototype was developed to give an example on how to use the SDE. Lets see how the new editor reacts with the same prototype. Particularly of interest is how the editor reacts when entering the timing constraints. For that reason, Figure 36 is repeated below.

---

**Figure 50. Unparsing Rules for operator_impl Phylum.**
Figure 36. Complete PSDL Program with Control Constraints (Repeated).

The same constraints will be inserted into the prototype just created. Notice in Figure 51 that when inputting the PER for operator Evaluate_Temp, a message comes up and states that if left unspecified, FW will default to PER. It will do this for PER, FW, MRT, and MCP.

Figure 51. SDE with Default Message.
Going further, the PER and FW is input for the Evaluate_Temp operator. So far, so good. The editor does not complain, so the Sensor operator is given the PER constraint defined earlier. As soon as the return key is hit, as Figure 52 indicates, the editor gives the familiar default message, but there is another message. The new scheduling notice states that the PER of the producer of a stream must exceed that of the consumer. This constraint, under the old editor, would have gone unnoticed until translated and scheduled, thereby eating up a great deal of time. Not to mention the fact that the designer may no longer have this information fresh in his or her memory.

![Figure 52. SDE with PER Constraint Error.](image)

But wait, there is another message. This prototype, because of the way the constraints were assigned will not run on a uniprocessor. The message at the top of the editor, based on the Load Factor discussed earlier, states that the prototype will require at least 2 processors to schedule. Apparently each operator was constrained in such a way that they both will require a processor of their own. Figure 53 shows the message at the top of the editor. This illustrates a very good point made earlier. As discussed in the beginning of this thesis, scheduling hard real-time systems is very difficult. If this system were real and had to run on a single CPU, we
would have already blown it. The designer needs quick feedback as to whether the design will work or not. If this isn’t provided, a great deal of time could be spent on a design that is impossible to implement, before the designer realizes it can’t work. The other semantic checks display error messages such as the ones shown thus far. All of the checks outlined in Table 4 & 5 earlier have been implemented and their message reads very much like the remarks of the tables. It is up to the reader to experiment with the SDE to see all of them for one self.

Figure 53. SDE with number processors required message.
V. FUTURE RESEARCH AND DEVELOPMENT

One of the biggest obstacles to new tools is acceptance by new, potential users. The learning curve plays a major part of this. If the tool is too difficult to use, no matter how good it is, most will reject it. For this reason, more and more time is being spent on developing good user interfaces as well as developing good tools.

Learning how to use the SDE not a trivial task and many are intimidated by it at first. The graphical editor, however, comes somewhat natural to most users. The CAPS tool-set would be embraced more strongly if the input functionality was moved to the graphical editor. Users would not have to learn to use a new editor nor would they have to learn a new language (PSDL). Information could be taken from and presented to the designer in a more intuitive fashion.

The information that is currently input via the graphical editor is propagated to the SDE. Research should to be done to determine how much more of this propagation can be achieved going in both directions between the derivation tree of the SDE and the graphical editor. The goal should be such that not only will the Graphical Editor account for all (or as much as possible) of the input, but well formed messages should be displayed back to the designer while still in the Graphical Editor. A separate window, perhaps utilizing TAE+, could be designed for displaying both the feedback messages that currently exist within the SDE and those of future implementations.

The display of how many processors are required to schedule the prototype can be particularly useful to the designer. Because the SDE can not convert the sporadic operators to have an equivalent period, it is not as accurate as could be. Some research should be devoted to whether or not this can be effectively done within the editor.

Additional checking that could be accomplished with the attributes already gathered include testing for constrained operators that have unconstrained children. Also, using the edges and operator MET’s, each thread under a parent operator could be checked to ensure its total required execution time does not exceed that of the parent.

Other checking that can be accomplished is the verification of streams that are built during decomposition. For example if a parent operator has an external stream coming in and two regular streams coming out, when it is decomposed, the graph editor shows those
connections at the bottom as a reminder. If the designer doesn’t include them or puts additional incoming or outgoing streams in, there are no warnings upon returning to the SDE. This inconsistency should not be allowed and can be tracked with the current edge attributes captured.

Many of the templates displayed in the help pane (menu window) should not be there. For example, the designer could pick psdl_implementation out of the help pane to replace the placeholder <operator implementation>. This does not work well and really shouldn’t be an option to the designer. If the operator is to be implemented with PSDL, that should be done within the graphical editor. The same holds for input and output stream. It is more confusing to the new users of the SDE to have those displayed. Research should be done on what parts of the display can be removed for a more clear implementation of the editor.
LIST OF REFERENCES


APPENDIX A - PSDL Grammar

psdl
  = (component)
component
  = data_type
    | operator
  
data_type
  = "type" id type_spec type_impl
  
type_spec
  = "specification" ["generic" type_decl] [type_decl]
    ("operator" op_name operator_spec)
    [functionality] "end"
  
operator
  = "operator" op_name operator_spec operator_impl
  
operator_spec
  = "specification" (interface) [functionality] "end"
  
interface
  = attribute [reqts_trace]
  
attribute
  = "generic" type_decl
    | "input" type_decl
    | "output" type_decl
    | "states" type_decl "initially" initial_expression_list
    | "exceptions" id_list
    | "maximum execution time" time
  
type_decl
  = id_list [":" type_name ("," id_list [":" type_name)]
  
type_name
  = id
    | id ["*" type_decl ""]
  
id_list
  = id ("," id)
  
reqts_trace
  = "required by" id_list
  
functionality
  = [keywords] [informal_desc] [formal_desc]
  
keywords
  = "keywords" id_list
  
informal_desc
  = "description" "(" text ")"
  
formal_desc
  = "axioms" "(" text ")"
  
type_impl
  = "implementation ada" id "end"
    | "implementation" type_name ("operator" op_name operator_impl) "end"
  
operator_impl
  = "implementation ada" ada_op_name "end"

| "implementation" psdl_impl "end"
  
psdl_impl
  = data_flow_diagram [streams] [timers] [control_constraints]
    [informal_desc]
  
data_flow_diagram
  = "graph" (vertex) (edge)
  
vertex
  = "vertex" op_id [":" time]
    -- time is the maximum execution time
  
edge
  = "edge" id [":" time] op_id "" op_id
    -- time is the latency
  
op_id
  = [id "]" op_name ["(" id_list ["" id_list [")"]"
  
streams
  = "data stream" type_decl
  
timers
  = "timer" id_list
  
control_constraints
  = "control constraints" constraint (constraint)
  
constraint
  = "operator" op_id
    ["triggered" [trigger] ["if" expression] [reqts_trace]]
    ["period" time [reqts_trace]]
    ["finish within" time [reqts_trace]]
    ["minimum calling period" time [reqts_trace]]
    ["maximum response time" time [reqts_trace]]
    [constraint_options]
  
constraint_options
  = "output" id_list ["if" expression [reqts_trace]]
    | "exception" id ["if" expression [reqts_trace]]
    | timer_op id ["if" expression [reqts_trace]]
  
trigger
  = "by all" id_list
    | "by some" id_list
  
timer_op
  = "reset timer"
    | "start timer"
    | "stop timer"
  
initial_expression_list
  = initial_expression ("," initial_expression)
  
initial_expression
  = "true"
    | "false"
    | integer_literal
    | real_literal
    | string_literal
    | id
    | type_name ":" op_name ["(" initial_expression_list [")"]"
APPENDIX A - PSDL Grammar

| "(" initial_expression ")" |
| initial_expression binary_op initial_expression |
| unary_op initial_expression |

binary_op
= "and" | "or" | "xor" |
| "<" | ">" | "<=" | ">=" |
| "*" | "/" | "%" | "mod" | "rem" | "**" |

unary_op
= "not" | "abs" | "-" | "~" |

time
= integer_literal unit

unit
= "microsec" |
| "ms" |
| "sec" |
| "min" |
| "hours" |

expression_list
= expression ("," expression)

expression
= "true" |
| "false" |
| integer_literal |
| time |
| real_literal |
| string_literal |
| id |
| type_name "." op_name ("(" expression_list ")") |
| ("(" expression ")") |
| ("(" expression ")") |
| initial_expression binary_op initial_expression |
| unary_op initial_expression |

op_name
= ada_op_name "." integer_literal

ada_op_name
= id "." integer_literal

id
= letter (alpha_numeric)

real_literal
= integer_literal "." integer_literal

integer_literal
= digit (digit)

string_literal
= """ (char) ...""

char
= any printable character except """"

digit
= "0 .. 9" |

letter
APPENDIX B - Abstract Rules

root prototype;

/* TOKEN DEFINITIONS */
CLINEBREAK: Clinebreak<NO_WHITESPACE>[\n] >;
CLINE: ClineLex <NO_WHITESPACE> ['\n'] >;
LCURLY: LCurly<{}<NO_WHITESPACE>> >;
RCURLY: RCurly<{}<INITIAL>> >;
WHITESPACE: WhitespaceLex<\s\S\n> >;

/* -------------------------*/

DESCRIPTION: DescripWxLex"DESCRIPTION" >
    |<"description">
INPUT: <INPUT> |
    |<"Input">
OUTPUT: <OUTPUT> |
    |<"output">
EXCEPTIONS: <EXCEPTIONS> |
    |<"exceptions">
EXCEPTION: <EXCEPTION> |
    |<"exception">
AXIOMS: <AXIOMS> |
    |<"axioms">
ADA: <ADA>
    |<"ADA">
INTEGER: <INTEGER> |
    |<"integer">
REAL: <REAL> |
    |<"real">
BOOLEAN: <BOOLEAN> |
    |<"boolean">
TRUE: <TRUE> |
    |<"true">
FALSE: <FALSE> |
    |<"false">
MICRO: <MICRO> |
    |<"micro">
MS: MSLex<MS> |
    |<"ms">
SEC: SECLex<SEC> |
    |<"sec">
MIN: MINLex<MIN> |
    |<"min">
HOURS: HOURSLex<HOURS> |
    |<"hours">
NOT: NOTLex<NOT> |
    |<"not">
AND: ANDLex<AND> |
    |<"and">
ABS: ABSLex<ABS> |
    |<"abs">
GET: GET Lex<GET> |
    |<"get">
LT: LTELex<LT> |
    |<"<">
NEQV: NEQVLex<NEQV> |
    |<"/=">
RENAME: RENAMELex<RENAME> |
    |<"rem">MOD: MODLex<MOD> |
    |<"mod">EXP: EXPLex<EXP> >;
APPENDIX B - Abstract Rules

```plaintext
STOPFN: STOPLex('STOP\ TIMER' >
|<"STOP\ timer" >)
QUOTEFN: QUOTELex('""' >
IDENTIFIER : IdentLex([a-zA-Z][a-zA-Z_0-9]*) >
INTBERS : IntegerLex([0-9]) >

/** STRING_LITERAL: StringLitLex("\")+ */

/** abstract grammar for comment lines to allow for free textual input */

list commentLines:
  commentLines
    : exported CommentLinesNil()
    | exported CommentLinesPair(commentLine commentLines)
    ;

  commentLine
    : exported CommentLineNil()
    ;

  /* input syntax */
  yCommentLine(synthesized commentLine a ;)
  yCommentLines(synthesized commentLines a ;)

  /* commentLine- yCommentLine.a */
  commentLines- <NO_WHITESPACE> yCommentLines.a ;

  yCommentLine
    ::= ()
      | (CLINE) ($$a = Remove_Leading_Blanks_From_String(CLINE); )
    ;

  yCommentLines
    ::= (yCommentLine)
      | (yCommentLine CLINEBREAK yCommentLines)
    | (yCommentLine yCommentLinesPair(yCommentLine.a, CommentLinesNil))
      | (yCommentLine yCommentLinesPair(yCommentLine.a, yCommentLines2.a))
    ;

  /* IF OPTIONAL COMMENTS ARE EVER WANTED IN PSDL, THE FOLLOWING
   DECLARATIONS SHOULD COME HANDY. REFERR TO REFS' AND
   TAYLOR'S BOOK, CHAPTER EIGHT FOR AN EXPLANATION OF
   HOW THIS WORKS. */

/***********************************************************/

optional optionalComment:
  optionalComment
    : OptionalCommentNil()
    | OptionalCommentPrompt()
    | OptionalComment(commentLines)
  ;

  yOptionalComment (synthesized optionalComment a ;)
  yOptionalComment
    ::= ()
      | (LCURLY yCommentLines RCURLY)
        yCommentLines.tail=CommentLinesNil;
      $$a = OptionalComment(yCommentLines.reversed);
```

96
APPENDIX B - Abstract Rules

optional list o_timing_info_list;
o_timing_info_list
  : TimeInfoNone()
  | TimeInfoPair(o_timing_info
               o_timing_info_list)
  ;
*/

optional o_timing_info;
o_timing_info
  : exported OptTimingInfoNone()
  | exported OptTimingInfoPrompt()
  | exported OptTimingInfo(time regmts_trace)
  ;
time
  : exported TimeNull()
  | exported Time(integer time_unit);
time_unit
  : exported UnitH1()
  | exported UnitMICROSECONDS()
  | exported UnitMS()
  | exported UnitSEC()
  | exported UnitMIN()
  | exported UnitHOURS();

optional list o_states_list;
o_states_list
  : exported StatesListNone()
  | exported StatesListPair(o_states
                           o_states_list);
o_states
  : exported OpStatesNone()
  | exported OpStates(type_declarations expression_list regmts_trace)
  ;

optional list initial_args;
/*
initial_args
  : InitialArgNil()
  ;
*/
initial_args
  : InitialArgPrompt()
  | InitialArg(expression_list)
  ;
an_argument
  : AnArgNil()
  | AnArgument(expression_list);

list expression_list;
expression_list
  : InitialExpListNil()
  | InitialExpListPair(expression expression_list)
  ;

expression
  : ExpNull()
  | Identifier(id)
  | Textual_Description(commentLines)
  ;
/*
  | Textual_Description(string_lit)
  */
  | TypeExpression(type_name id initial_args)
  | ParenthesizedExp(expression)
  ;

  /* BOOLEAN EXPRESSIONS */
  | True()
  | False()
  | NotExp(expression)
  | EqualExp(expression expression)
  | LessExp(expression expression)
  | GreaterExp(expression expression)
  | GreatEqualExp(expression expression)
  | LessEqualExp(expression expression)
  | NotEqualExp(expression expression)
  | AndExp(expression expression)
  | OrExp(expression expression)
  | XorExp(expression expression)
  ;

  /* ARITHMETIC EXPRESSIONS */
  | Integer(integer)
  | Real(integer integer)
  | PlusExp(expression expression)
  | MinusExp(expression expression)
  | TimesExp(expression expression)
  | DivExp(expression expression)
  | NegativeExp(expression)
  | PositiveExp(expression)
  | AbsExp(expression)
  | RemExp(expression expression)
  | ModExp(expression expression)
  | ExponentExp(expression expression)
  ;

  /* STRING EXPRESSION */
  | ConcatExp(expression expression)
  ;

optional o_keywords;
o_keywords
  : KeyWordsNone()
  | KeyWordsPrompt()
  | KeyWords(alone_id_list)
  ;

optional o_informal_descs;
o_informal_descs
  : exported InformalDescsNull()
  | exported InformalDescsPrompt()
  | exported InformalDescs(commentLines)
  ;

optional o_formal_descs;
o_formal_descs
  : FormalDescsNone()
  | FormalDescsPrompt()
  | FormalDescs(commentLines)
  ;
optional regmats_trace;
    regmats_trace
    
        : RegmatsTraceNone()
        | RegmatsTracePrompt()
        | RegmatsTrace(alone_id_list)
        ;

    text
    
        : TextNull()
        | Text(id)
        ;

type_impl
    
        : TypeImplNull()
        | AdaTypeImpl(id)
        | TypeImpl(type_name
                     operator_impl_list)
        ;

    /* new declarations */

    optional list operator_impl_list;
    operator_impl_list
    
        : OpImplListNull()
        | OpImplListPair(t_op_impl
                        operator_impl_list)
        ;

t_op_impl
    
        : TOpImplNull()
        | TOpImpl(id
                 operator_impl)
        ;

    operator_impl
    
        : exported OpImplNull()
        | exported AdaOpImpl(id)
        | exported OperatorImpl(graph
                                 declarations
                                 cc)
        ;

    graph
    
        : exported GraphNull()
        | exported Graph(vertex_list
                         edge_list)
        ;

    optional list vertex_list;
    vertex_list
    
        : exported VertexListNull()
        | exported VertexListPair(a_vertex
                                  vertex_list)
        ;

    a_vertex
    
        : exported AVertexNull()
        | exported AVertex(operator_id
                           optional_time)
        ;

    operator_id
    
        : exported OperatorIdNull()
        | exported OperatorId(optional_type_id
                              id
                              operator_id_pairs)
        ;

    optional optional_type_id;
    optional_type_id
    
        : exported OptOptionalIdNull()
        | exported OptOptionalIdPrompt()
        | exported OptOptionalId(id)
        ;

    optional operator_id_pairs;
    operator_id_pairs
    
        : exported OperatorIdPairsNull()
        | exported OperatorIdPairsPrompt()
        | exported OperatorIdPairs(alone_id_list
                                  alone_id_list)
        ;

    optional optional_time;
    optional_time
    
        : exported OptOptionalTimeNull()
        | exported OptOptionalTimePrompt()
        | exported OptOptionalTime(time)
        ;

    optional list edge_list;
    edge_list
    
        : exported EdgeListNull()
        | exported EdgeListPair(an_edge
                                 edge_list)
        ;

    an_edge
    
        : exported AnEdgeNull()
        | exported AnEdge(id
                         latency_time
                         from_vertex_id
                         to_vertex_id)
        ;

    optional latency_time;
    latency_time
    
        : exported LatencyTimeNull()
        | exported LatencyTimePrompt()
        | exported LatencyTime(time)
        ;

    from_vertex_id
    
        : FVertexIdNull()
        | FVertexId(optional_type_id
                               id
                               operator_id_pairs)
        ;

    to_vertex_id
    
        : TVertexIdNull()
        | TVertexId(optional_type_id
                    id
                    operator_id_pairs)
        ;
declarations
/
    : exported DeclarationsNull()
    | exported Declarations(optional_streams)
/
    : exported Declarations[optional_streams]
        optimal_timers
    :

    optional optional_streams;
    optional_streams
        : exported StreamsNull()
        | exported StreamsPrompt()
        | exported Streams(type_declarations)
    :

    optional optional_timers;
    optional_timers
        : exported TimersNull()
        | exported TimersPrompt()
        | exported Timers(alone_id_list)
        :

    cc
        : exported CcNull()
        | exported Cc(constraints
            o_informal_descs)
    :

    list constraints;
    constraints
        : exported ConstraintsNull()
        | exported ConstraintsPair(a_constraint
            constraints)
    :

    a_constraint
        : exported AConstraintNull()
        | exported AConstraint(operator_id
            optional_trigger
            optional_period
            optional_finish_within
            optional_mcp
            optional_mrt
            output Guards
            exception_ops
            timer_operations)
    :

    optional optional_trigger;
    optional_trigger
        : exported OptionalTriggerNull()
        | exported OptionalTriggerPrompt()
        | exported OptionalTriggerAllOrSome
            type_of_trigger
            alone_id_list
            optional_if_predicate
            regmets_trace
        | exported OptionalIfExp(expression
            regmets_trace)
    :

type_of_trigger
    : exported TriggerNull()
    | exported TriggerAll()
    | exported TriggerSome()
    :

    optional optional_period;
    optional_period
        : exported OptPeriodNull()
        | exported OptPeriodPrompt()
        | exported OptPeriod(time
            regmets_trace)
    :

    optional optional_finish_within;
    optional_finish_within
        : exported OptFinishWithinNull()
        | exported OptFinishWithinPrompt()
        | exported OptFinishWithin(time
            regmets_trace)
    :

    optional optional_mcp;
    optional_mcp
        : exported OptMcpNull()
        | exported OptMcpPrompt()
        | exported OptMcp(time regmets_trace)
    :

    optional optional_mrt;
    optional_mrt
        : exported OptMrtNull()
        | exported OptMrtPrompt()
        | exported OptMrt(time regmets_trace)
    :

    optional list output_guards;
    output_guards
        : exported Output GuardsNull()
        | exported Output GuardsPair(a_guard output_guards)
    :

    a_guard
        : exported AGuardNull()
        | exported AGuard(alone_id_list
            c_expression
            regmets_trace)
    :

    optional exception_ops;
    exception_ops
        : exported ExceptionOpsNull()
        | exported ExceptionOpsPrompt()
        | exported Exception(exception_options)
    :

    optional list exception_options;
    exception_options
        : exported ExceptionOptionsNull()
        | exported ExceptionOptionsPair(an_exception
            exception_options)
    :

    an_exception
APPENDIX B - Abstract Rules

```c

/*
  expression
  = expression op expression
  | expression rel expression
  | expression op expression
  | expression
  */

expression op expression
  = expression op expression
  | expression rel expression
  | expression op expression
  | expression

expression op expression
  = expression op expression
  | expression rel expression
  | expression op expression
  | expression

expression rel expression
  = expression rel expression
  | expression rel expression
  | expression rel expression
  | expression

expression
  = expression
  | expression

op expression
  = expression op expression
  | expression op expression
  | expression op expression
  | expression

rel expression
  = expression rel expression
  | expression rel expression
  | expression rel expression
  | expression
```

---

The above text is a part of the syntax for expressions in a programming language, likely C, given the syntax structure and terminology used. The document is part of an appendix, suggesting it contains information related to rules or constraints used in the context described.

---

### Notes

- **Expression Syntax**
  - `expression` can be an operand, an operator (`op`), or a comparison (`rel`).
  - Operators can be binary or unary.
  - Comparisons include equality, inequality, greater than, etc.
  - Expressions can be concatenated with `expression list`.

- **Example Usage**
  - `op expression op expression`
  - `expression rel expression`
  - `expression`

---

**Additional Context**

This syntax is likely part of a larger documentation, possibly for a compiler or interpreter, detailing the abstract syntax tree (AST) for expressions in a particular language.
APPENDIX B - Abstract Rules

/* Maximum execution time * /
/* Due to problems when working w/ time, all constraint times will be 
converted to a common integer (assume microseconds) */
met
  : exported MetNull()
  | exported Met(REAL)

/*-----------------------------------------------*/

list id_constraint_set;

id_constraint_set
  : exported IdConstraintSetNil()
  | exported IdConstraintPair(id_constraint id_constraint_set)

id_constraint
  : exported IdConstraintNull()
  | exported IdConstraint(operator_id time_constraint trigger)

time_constraints
  : exported OpConstraintNull()
  | exported Periodic(per fw)
  | exported Sporadic(mcp mrt)

trigger
  : exported TriggerByNull()
  | exported TriggerByAll()
  | exported TriggerBySome()

/* Period */
per
  : exported PerNull()
  | exported Per(REAL)

/* Finish Within */
fw
  : exported FWNull()
  | exported FW(REAL)

/* Minimum calling period */
mcp
  : exported McpNull()
  | exported Mcp(REAL)

/* Maximum response time */
mrt
  : exported MrtNull()
  | exported Mrt(REAL)

/*-----------------------------------------------*/

list edge_set;

edge_set

edge
  : exported EdgeNull()
  | exported Edge(edge edge_set)

latency
  : exported LatencyNull()
  | exported Latency(REAL)

producer
  : exported ProducerNull()
  | exported Producer(operator_id)

consumer
  : exported ConsumerNull()
  | exported Consumer(operator_id)
APPENDIX C - Attribute Rules

```plaintext
prototype, pdsl_components
{
    syn pdsl_components syn_defined_operators;
    syn pdsl_components syn_defined_types;

    /*
    syn type_declarations syn_defined_streams;
    syn type_declarations syn_defined_states;
    */
    syn type_declarations syn_defined_type_decl;
    syn vertex_list syn_defined_vertices;
    /*
    syn edge_list syn_defined_edges;
    */
};

/*****************************************************************************/

prototype, pdsl_components, component (syn op_id_met_set syn_vertex_id_met_set);
prototype, pdsl_components, component (syn id_constraint_set syn_id_constraint_set);

prototype
{
    syn IdSet syn_root_ids;
    syn IdSet syn_multiple_root_ids;
    syn IdSet syn_multiple_op_spec;
    syn IdSet syn_multiple_type_spec;
    syn OpIdSet syn_multiple_vertices;
    syn IdSet syn_multiple_type_streams;
};

prototype : Prot {
    $$syn_vertex_id_met_set = pdsl_components.syn_vertex_id_met_set;
    $$syn_id_constraint_set = pdsl_components.syn_id_constraint_set;
    $$syn_defined_operators = pdsl_components.syn_defined_operators;
    $$syn_defined_types = pdsl_components.syn_defined_types;

    $$syn_defined_type_decl = pdsl_components.syn_defined_type_decl;
    $$syn_defined_vertices = pdsl_components.syn_defined_vertices;

    local IdSet root_components;
    root_components = Extract_Root Components(
        pdsl_components.syn_defined_operators,
        pdsl_components.syn_defined_vertices);
    $$syn_root_ids = root_components;

    local BOOL multiple_root_error;
    multiple_root_error = (IdSetSize(root_components) > 1);

    local STR multiple_root_message;
    multiple_root_message = (multiple_root_error)
    ? "\n    \n    Multiple Root Operators \n"
    \n    ;

    local IdSet multiple_root_ids;
    multiple_root_ids = (multiple_root_error)
    ? root_components :
        IdSetNull;
    $$syn_multiple_root_ids = multiple_root_ids;

    local IdSet multiple_op_spec;
    multiple_op_spec = Extract_Multiple_Op_Spec_Id(
        pdsl_components.syn_defined_operators);
    $$syn_multiple_op_spec = multiple_op_spec;

    local BOOL multiple_op_spec_error;
    multiple_op_spec_error = (!$isNull(multiple_op_spec))?
        "\n        \n        Components With More Than 1 Operator Spec
"
        ;

    local IdSet multiple_type_spec;
    multiple_type_spec = Extract_Multiple_Type_Spec_Id(
        pdsl_components.syn_defined_types);
    $$syn_multiple_type_spec = multiple_type_spec;

    local BOOL multiple_type_spec_error;
    multiple_type_spec_error = (!$isNull(multiple_type_spec))?
        "\n        \n        Components With More Than 1 Type Spec
"
        ;

    local STR multiple_type_spec_message;
    multiple_type_spec_message = (multiple_type_spec_error)
    ? "\n    \n    Components Defined As Both Operators and Types
"
    \n    ;

    local IdSet also_op_type_ids;
    also_op_type_ids = Extract_Op_Type_Spec_Id(
        pdsl_components.syn_defined_types,
        pdsl_components.syn_defined_operators);

    local BOOL also_op_type_error;
    also_op_type_error = (!$isNull(also_op_type_ids))?
        "\n        \n        Components Defined As Both Operators and Types
"
        ;

    local STR also_op_type_message;
    also_op_type_message = (also_op_type_error)
    ? "\n    \n    Components Defined As Both Operators and Types
"
    \n    ;

    local IdSet undefined_op_spec_set;
    undefined_op_spec_set = Extract_Undefined_Op(
        pdsl_components.syn_defined_operators,
        pdsl_components.syn_defined_operators);

    local BOOL undefined_op_spec_error;
    undefined_op_spec_error = (!$isNull(undefined_op_spec_set))?
        "\n        \n        Operators Missing Operator Spec
"
        ;

    local STR undefined_op_spec_message;
    undefined_op_spec_message = (undefined_op_spec_error)
    ? "\n    \n    Operators Missing Operator Spec
"
    \n    ;

    local OpIdSet undefined_type_op_spec_set;
    undefined_type_op_spec_set = Extract_Undefined_Type_Op(
        pdsl_components.syn_defined_vertices,
        pdsl_components.syn_defined_types);

    local BOOL undefined_type_op_spec_error;
    undefined_type_op_spec_error = (!$OpIdSet$isNull(undefined_type_op_spec_set))?
        "\n        \n        Operators Missing Type and Op Spec
"
        ;

    local STR undefined_type_op_spec_message;
    undefined_type_op_spec_message = (undefined_type_op_spec_error)
    ? "\n    \n    Operators Missing Type and Op Spec
"
    \n    ;

    103
```
local OpIDSet multiple_vertices;
multiple_vertices = Extract_Multiple_Vertices(
    psdl_components.syn_defined_vertices);

$n$.syn_multiple_vertices = multiple_vertices;

local BOOL multiple_vertices_error;
multiple_vertices_error = (!OpIDSetNullOrEmpty(multiple_vertices));

local STR multiple_vertices_message;
multiple_vertices_message = (multiple_vertices_error)
    ? "$\n\n-- Vertices Appeared In More Than 1 Operator Impl$" : "$\n\n-- Vertices Appeared In More Than 1 Operator Impl$";

local IdSet multiple_streams;
multiple_streams = Extract_Multiple_TD(psdl_components.syn_defined_type_decl);
$n$.syn_multiple_streams = multiple_streams;

local BOOL multiple_streams_error;
multiple_streams_error = (!IdSetNullOrEmpty(multiple_streams));

local STR multiple_streams_message;
multiple_streams_message = (multiple_streams_error)
    ? "$\n\n-- Streams With More Than 1 State/Data Stream Declaration$" : "$\n\n-- Streams With More Than 1 State/Data Stream Declaration$";

local INT processor_number;
processor_number = Min_Processors_Required($n$.syn_vertex_id_met_set,
    $n$.syn_id_constraint_set);

local BOOL min_processor_required_GT1;
min_processor_required_GT1 =
    (1 < processor_number);

local STR min_processor_msg;
min_processor_msg =
    (min_processor_required_GT1)
    ? "$\n\n-- Prototype will not schedule for less than$" # "\n\n-- Prototype will not schedule for less than$"
    INT2STR(processor_number)
    # processors. (Based on Periodic Operators only)$" : "$\n\n-- Prototype will not schedule for less than$" # "\n\n-- Prototype will not schedule for less than$"
    # processors. (Based on Periodic Operators only)$";

local BOOL uniprocessor_unschedulability_error;
uniprocessor_unschedulability_error =
    !min_processor_required_GT1 &&
    !Uniprocessor_Schedulable($n$.syn_vertex_id_met_set,
    $n$.syn_id_constraint_set);

local STR uniprocessor_unschedulability_msg;
uniprocessor_unschedulability_msg =
    (uniprocessor_unschedulability_error)
    ? "$\n\n-- Prototype will not schedule for Uniprocessor.$\n\n-- Prototype will not schedule for Uniprocessor.$",
    # Based on Periodic Operators only)$" : "$\n\n-- Prototype will not schedule for Uniprocessor.$\n\n-- Prototype will not schedule for Uniprocessor.$",
    # Based on Periodic Operators only)$";

local BOOL has_error;
has_error = (multiple_root_error
    || multiple_op_spec_error
    || multiple_type_spec_error
    || also_op_type_error
    || undefined_op_spec_error
    || undefined_type_op_spec_error
    || multiple_vertices_error
    || multiple_streams_error
    || min_processor_required_GT1
    || uniprocessor_unschedulability_error);

local STR error_header;
error_header = (has_error)
    ? "$\n\n-- WARNINGS, ERRORS AND ALERTS$" : "$\n\n-- WARNINGS, ERRORS AND ALERTS$";

local STR error_trailer;
error_trailer = (has_error)
    ? "$\n\n-- WARNINGS, ERRORS AND ALERTS$" : "$\n\n-- WARNINGS, ERRORS AND ALERTS$";

store(default_store global_proto_store) local prototype global_proto;
global_proto = prototype;

store(default_store global_root_store) local IdSet global_root;
global_root = root_components;

store(default_store global_type_decl_store) local type_declarations
    global_type_decl = psdl_components.syn_defined_type_decl;

store(default_store global_undef_ops_store) local IdSet global_undef_ops;
global_undef_ops = undefined_op_spec_set;

/--------------------------------------------------------------------------*/

psdl_components
    : PadM1
      ($$.syn_defined_operators = PadM1;
        $$.syn_defined_types = PadM1;
        $$.syn_defined_type_decl = TypeDecM1;
        $$.syn_defined_vertices = VertexListM1;
        $$.syn_vertex_id_met_set = CmplSortSetM1;
        $$.syn_id_constraint_set = IdConstraintSetM1;
      )
    | PadPair
      ($$.syn_defined_operators =
        (IoOperator(component)
          ? GetOperator(component):;
            psdl_components$1.syn_defined_operators
            : psdl_components$2.syn_defined_operators)
          : $$.syn_defined_types
          = (IoType(component)
            ? GetType(component):;
              psdl_components$1.syn_defined_types
              : psdl_components$2.syn_defined_types)
          : $$.syn_defined_type_decl =
            Concat_Type_Decl_List(
            Concat_Type_Decl_List(
            )
          );

104
APPENDIX C - Attribute Rules

```c
Concat_Type_Decl_List(
    Get_States(component),
    Get_Streams(component),
    (IfType(component)
    ? Concat_Type_Decl_List(
        Get_Inputs(component),
        Get_Outputs(component))
    : TypeDeclNil))

psdl_components$2.syn_defined_type_decl); 

$$._syn_defined_vertices =
    Concat_Vertex_List(
    GetVertices(component),
    psdl_components$2.syn_defined_vertices);

$$._syn_vertex_id_met_set =
    Op_Id_Meta_Set_Union(component.syn_vertex_id_met_set,
    psdl_components$2.syn_vertex_id_met_set);

$$._syn_id_constraint_set =
    Id_Constraint_Set_Union(component.syn_id_constraint_set,
    psdl_components$2.syn_id_constraint_set);

/*******************************************************************************/

$2.type_declarations, a.decl (synthesized IdSet id_set); 

id_list (inh type_declarations inh_types_types;
    inh type_declarations inh_generic_types;
    inh type_declarations inh_generic_types;
    inh type_declarations inh_outputs_types;
    inh type_declarations inh_states_types;
);

id_list
  : IN Nil
    {$$.id_set = IdSetNil;
    }
  | IdPair
    {$$.id_set = IdSetUnion(SingletonIdSet(id), id_list$2.1.id_set);
    id_list$2.1.inh_types_types = $$.inh_types_types;
    id_list$2.1.inh_generic_types = $$.inh_generic_types;
    id_list$2.1.inh_generic_types = $$.inh_generic_types;
    id_list$2.1.inh_outputs_types = $$.inh_outputs_types;
    id_list$2.1.inh_states_types = $$.inh_states_types;
    };

id_list:
  IdPair
    (local STR multiply_defined;
     multiply_defined =
       ((id != IdNull) &
       (IdDefInType(id, $$.inh_types_types) +
       IdDefInType(id, $$.inh_generic_types) +
       IdDefInType(id, $$.inh_generic_types) +
       IdDefInType(id, $$.inh_inputs_types) +
       IdDefInType(id, $$.inh_outputs_types) +
       IdDefInType(id, $$.inh_states_types) > 1))
```
APPENDIX C - Attribute Rules

0;

}$$.generic_types = o.generic_params.declared_types;
$$$.types_types = o_type_decls.declared_types;
$$.defined_operators = o_operators;

o.generic_params.inh_types_types = $$$.types_types;
o_type_decls.inh_types_types = $$$.types_types;
o.generic_params.inh_generic_types = $$$.generic_types;
o_type_decls.inh_generic_types = $$$.generic_types;

/**------------------------------------------*/
type_impl( syn operator_impl_list syn_operator_impl_list;

$$$.syn_operator_impl_list = OpImplListNull;
$$$.syn_operator_impl_list = OpImplListNull;
$$$.syn_operator_impl_list = operator_impl_list;

/**------------------------------------------*/
operator_spec, o_inputs_list, o_inputs ( syn IdSet input_id_set;
operator_spec, o_outputs_list, o_outputs ( synthesized IdSet output_id_set;
operator_spec, o_states_list, o_states ( synthesized IdSet state_id_set;

operator_impl, graph, vertex_list, a_vertex, operator_id ( synthesized OpIdSet vertex_id_set;
operator_impl, graph, vertex_list ( syn_op_id_met_set syn_vertex_id_met_set;
operator_impl, graph, edge_list ( syn edge_set syn_edge_set;
operator_impl, cc, constraints ( syn_id_constraint_set syn_id_constraint_set;

cc, constraints, a_constraint ( inh_op_id_met_set inh_vertex_id_met_set;

operator_impl, graph, edge_list, an_edge ( synthesized IdSet edge_id_set;
operator_impl, declarations, optional_streams ( synthesized IdSet stream_id_set;

operator_impl, cc, constraints, a_constraint ( synthesized OpIdSet constraint_op_id_set;

o.generics_list (syn type_declarations declared_types);

o.generics_list, o.generics

{ inh type_declarations inh_generic_types;
inh type_declarations inh_inputs_types;
in type_declarations inh_outputs_types;
in type_declarations inh_states_types;
}

o.generics_list (GenericsListNone (declared_types = TypeDecNil;

| GenericsListPair

106
APPENDIX C - Attribute Rules

```plaintext
{type_declarations.inh_types_types = TypeDeclNil;
 type_declarations.inh_generic_types = TypeDeclNil;
 type_declarations.inh_generics_types = $$.inh_generics_types;
 type_declarations.inh_inputs_types = $$inha_inputs_types;
 type_declarations.inh_outputs_types = $$inha_outputs_types;
 type_declarations.inh_states_types = $$inha_states_types;}

/*--------------------------------------------------------------------------*/
on_outputs_list (syn type_declarations declared_types);}

on_outputs_list, o_outputs
{| 
    inh_type_declarations inh_generics_types;
    inh_type_declarations inh_inputs_types;
    inh_type_declarations inh_outputs_types;
    inh_type_declarations inh_states_types;
};

on_outputs_list
{| OutputListNone
    ($.declared_types = TypeDeclNil;
     $.output_id_set = IdSetNil;
    )
    | OutputListPair
        ($$.declared_types = GetOutputsTypes(o_outputs); o_outputs_list$$.declared_types;
         $.output_id_set = IdSetUnion(o_outputs_list$$.output_id_set, o_outputs_list$$.input_id_set);
        )
        | InputsListNone
            ($$.declared_types = GetInputsTypes(o_inputs); o_inputs_list$$.declared_types;
             $.input_id_set = IdSetNil;
            )
            | InputsListPair
                ($$.declared_types = GetInputsTypes(o_inputs); o_inputs_list$$.declared_types;
                 $.input_id_set = IdSetUnion(o_inputs_list$$.input_id_set, o_inputs_list$$.input_id_set);
                )
        | OOutputs
            ($$.output_id_set = IdSetNil;
            )
            | OInputNone
                ($$.input_id_set = IdSetNil;
                )
        | OOutputs
            ($$.output_id_set = type_declarations.id_set;
            )
            | OInputs
                ($$.input_id_set = type_declarations.id_set;
                )
        | OStates
            ($$.states_id_set = type_declarations.id_set;
            )
            | OStates
                ($$.states_id_set = type_declarations.id_set;
                )

/*--------------------------------------------------------------------------*/
on_states_list (syn type_declarations declared_types);}

on_states_list, o_states
{| 
    inh_type_declarations inh_generics_types;
};
```
APPENDIX C - Attribute Rules

inh type_declarations inh_inputs_types;
inherited_type_declarations inh_outputs_types;
inherited_type_declarations inh_states_types;
);

o_states_list
: StatesListNone
{$$,declared_types = TypeDeclNil;
 $$,state_id_set = IdSetNil;
 }
 | StatesListPair
{$$,declared_types = GetStatesTypes(o_states) @ o_states_list$2,declared_types;
 $$,state_id_set = IdSetUnion(o_states.state_id_set, o_states_list$2.state_id_set);
 o_states_list$2.inh_generics_types = $$,generics_types;
 o_states_list$2.inh_inputs_types = $$,inh_inputs_types;
 o_states_list$2.inh_outputs_types = $$,inh_outputs_types;
 o_states_list$2.inh_states_types = $$,inh_states_types;
 o_states.inh_generics_types = $$,inh_generics_types;
 o_states.inh_inputs_types = $$,inh_inputs_types;
 o_states.inh_outputs_types = $$,inh_outputs_types;
 o_states.inh_states_types = $$,inh_states_types;
 $$,syn_met = o.timing_info.syn_met;
 );

o_states
: OpStatesNone
{$$,state_id_set = IdSetNil;
 }
 | OpStates
{$$,state_id_set = type_declarations.id_set;
 type_declarations.inh_types_types = TypeDeclNil;
 type_declarations.inh_generics_types = TypeDeclNil;
 type_declarations.inh_inputs_types = $$,inh_inputs_types;
 type_declarations.inh_outputs_types = $$,inh_outputs_types;
 type_declarations.inh_states_types = $$,inh_states_types;
 }

/*--------------------------------------*/
o_timing_info
{syn time syn_met; }

o_timing_info
: OpTimingInfoNone( $$,syn_met = TimeNull; )
 | OpTimingInfoPrompt( $$,syn_met = TimeNull; )
 | OpTimingInfo( $$,syn_met = time; );

/*--------------------------------------*/
operator_spec
{syn type_declarations generics_types;
 syn type_declarations inputs_types;
 syn type_declarations outputs_types;
 syn type_declarations states_types;
 syn time syn_met; }

operator_spec
: OperatorSpec(
 $$,generics_types = o.generics_list.declared_types;
 $$,inputs_types = o.inputs_list.declared_types;
 $$,outputs_types = o.outputs_list.declared_types;
 $$,states_types = o.states_list.declared_types;
 $$,id_set = o.inputs_list.input_id_set;
 $$,output_id_set = o.outputs_list.output_id_set;
 $$,state_id_set = o_states_list.state_id_set;
 o.generics_list.inh_generics_types = $$,generics_types;
 o.generics_list.inh_inputs_types = $$,inputs_types;
 o.generics_list.inh_outputs_types = $$,outputs_types;
 o.generics_list.inh_states_types = $$,states_types;
 o.inputs_list.inh_generics_types = $$,generics_types;
 o.inputs_list.inh_inputs_types = $$,inputs_types;
 o.inputs_list.inh_outputs_types = $$,outputs_types;
 o.inputs_list.inh_states_types = $$,states_types;
 o.outputs_list.inh_generics_types = $$,generics_types;
 o.outputs_list.inh_inputs_types = $$,inputs_types;
 o.outputs_list.inh_outputs_types = $$,outputs_types;
 o.outputs_list.inh_states_types = $$,states_types;
 o.states_list.inh_generics_types = $$,generics_types;
 o.states_list.inh_inputs_types = $$,inputs_types;
 o.states_list.inh_outputs_types = $$,outputs_types;
 o.states_list.inh_states_types = $$,states_types;
 $$,syn_met = o.timing_info.syn_met;
 );

*/

/*--------------------------------------*/
  t_oper_spec
  : TopSpec(
    local BOOL multiply_defined_error;
    multiply_defined_error = (id! = IDNull) & (OpIsDefinedInTypeSpec( id, (type_spec,defined_operators)) > 1));
    local STR multiply_defined_message;
    multiply_defined_message = (multiply_defined_error)
      ? "\n-- \n-- Multiply Declared Operator Spec"
      : "";
    local BOOL has_error;
    has_error = (multiply_defined_error );
    local STR error_message;
    error_message = (has_error)
      ? "\n-- Warnings, Errors and Alerts:\n"      : "";
    local STR error_trailer;
    error_trailer = (has_error)
      ? "\n-- "      : "";
    store(default_error t_multi_op_error_store) local BOOL local_multi_op_error;
    local_multi_op_error = multiply_defined_error;
  );

108
APPENDIX C - Attribute Rules

/*------------------------------------------*/

t_op_impl :
TOpImpl
{
local t_op_impl local_t_op_impl;
local_t_op_impl = (id!=IdNull)
? Find_TOp_Spec(id, (Data.syn_o_operators))
: TOPSpec Nil;

local BOOL missing_op_spec_error;
missing_op_spec_error = (id!=IdNull)
& (local_t_op_impl == TOPSpec Nil);

local STR missing_op_spec_message;
missing_op_spec_message = (missing_op_spec_error)
? "$" " " \n-- \n-- Obsolete Operator Implementation\n"
: "$";

local BOOL multiply_defined_error;
multiply_defined_error = ((id!=IdNull) &&
(OpIsDefinedInTypeImpl(id,
(type_impl.syn_operator_impl_list) > 1)));

local STR multiply_defined_message;
multiply_defined_message = (multiply_defined_error)
? "$" " " \n-- \n-- Multiply Defined Operator Impl\n"
: "$";

local IdSet inh_input_id_set;
inh_input_id_set = Get_Inh_Id_Set_From_TOPSpec(local_t_op_impl);

local IdSet inh_output_id_set;
inh_output_id_set = Get_Inh_Output_Id_Set_From_TOPSpec(local_t_op_impl);

local IdSet inh_state_id_set;
inh_state_id_set = Get_Inh_State_Id_Set_From_TOPSpec(local_t_op_impl);

local time inh_set;
inh_set = Get_Inh_Set_From_TOPSpec(local_t_op_impl);

local IdSet input_output_state_union_set;
input_output_state_union_set =
IdSetUnion(inh_state_id_set,
IdSetUnion(inh_input_id_set, inh_output_id_set));

local IdSet obsolete_state_id_set;
obsolete_state_id_set = (Get_Impl_Form(operator_impl) == 2)
? IdSetDifference(inh_state_id_set,
Extract_Edge_Id_Set(operator_impl))
: IdSetNil;

local BOOL obsolete_state_error;
obsolete_state_error = (IdNull(obsolete_state_id_set));

local STR obsolete_state_message;
obsolete_state_message = (obsolete_state_error)
? "$" " " \n-- \n-- Obsolete State Declarations \n"
: "$";

local IdSet undefined_stream;
undefined_stream =
IdSetDifference(operator_impl.edge_id_set,
IdSetUnion(operator_impl.stream_id_set, input_output_state_union_set));

local IdSet obsolete_stream;
obsolete_stream =
IdSetDifference(operator_impl.stream_id_set, operator_impl.edge_id_set,
IdSetIntersect(operator_impl.stream_id_set, input_output_state_union_set));

local BOOL undefined_stream_error;
undefined_stream_error = (IdNull(undefined_stream));

local STR undefined_stream_message;
undefined_stream_message = (undefined_stream_error)
? "$" " " \n-- \n-- Missing Stream Declarations \n"
: "$";

local BOOL obsolete_stream_error;
obsolete_stream_error = (IdNull(obsolete_stream));

local STR obsolete_stream_message;
obsolete_stream_message = (obsolete_stream_error)
? "$" " " \n-- \n-- Obsolete Stream Declarations \n"
: "$";

local Id loc_operator_id;
loc_operator_id = id;

local OpIdSet undefined_constraint;
undefined_constraint = OpIdSetDifference(operator_impl.vertex_id_set,
operator_impl.constraint_op_id_set);

local OpIdSet obsolete_constraint;
obsolete_constraint =
OpIdSetDifference(operator_impl.constraint_op_id_set,
operator_impl.vertex_id_set);

local BOOL undefined_constraint_error;
undefined_constraint_error = (OpIdSetIsNull(undefined_constraint));

local STR undefined_constraint_message;
undefined_constraint_message = (undefined_constraint_error)
? "$" " " \n-- \n-- Missing Constraint Entries \n"
: "$";

local BOOL obsolete_constraint_error;
obsolete_constraint_error = (OpIdSetIsNull(obsolete_constraint));

local STR obsolete_constraint_message;
obsolete_constraint_message = (obsolete_constraint_error)
? "$" " " \n-- \n-- Obsolete Constraint Entries \n"
: "$";

local BOOL has_error;
has_error = (missing_op_spec_error ||
multiply_defined_error ||
obsolete_state_error ||
undefined_stream_error ||
obsolete_stream_error ||
APPENDIX C - Attribute Rules

undefined_constraint_error ||
obsolete_constraint_error);

local STR error_header;
error_header = (has_error)

? "\n----------------------------------\n--
WARNINGS, ERRORS AND ALERTS:*
:
**;

local STR error_trailer;
error_trailer = (has_error)

? "\n----------------------------------\n

store(default_store t_operator_id_store) local id operator_name;
operator_name = id;

store(default_store t_type_store) local id type_name;
type_name = (Data.local_type_id);

store(default_store t_states_id_store) local IdSet local_states_idset;
local_states_idset = inh_state_id_set;

store(default_store t_inh_input_id_store) local IdSet local_inh_input_id_set;
local_inh_input_id_set = inh_input_id_set;

store(default_store t_inh_output_id_store) local IdSet local_inh_output_id_set;
local_inh_output_id_set = inh_output_id_set;

store(default_store t_inh_met_store) local time local_inh_met;
local_inh_met = inh_met;

store(default_store t_impl_store) local INT is_composite;
is_composite = Get_Impl_Form(operator_impl);

store(default_store t_vertex_id_store) local OpIdSet local_vertex_idset;
local_vertex_idset = operator_impl.vertex_id_set;

store(default_store t_edge_id_store) local IdSet local_edge_idset;
local_edge_idset = operator_impl.edge_id_set;

store(default_store t_stream_error_store) local BOOL local_stream_error;
local_stream_error = (undefined_stream_error || obsolete_stream_error)

store(default_store t_constraint_error_store) local BOOL local_constraint_error;
local_constraint_error = (undefined_constraint_error || obsolete_constraint_error)

/*----------------------------------------*/

operator_impl

: OpImplNil, AdaOpImpl

{
$$.vertex_id_set = OpIdSetNil;
$$.edge_id_set = IdSetNil;
$$.stream_id_set = IdSetNil;
$$.constraint_op_id_set = OpIdSetNil;
$$.syn_vertex_id_met_set = OpIdMetSetNil;
$$.syn_edge_set = EdgeSetNil;
$$.syn_id_constraint_set = IdConstraintSetNil;
}

| OperatorImpl
{
$$.vertex_id_set = graph.vertex_id_set;
$$.edge_id_set = graph.edge_id_set;
$$.stream_id_set = declarations.stream_id_set;
$$.constraint_op_id_set = cc.constraint_op_id_set;
$$syn_vertex_id_met_set = graph.syn_vertex_id_met_set;
$$syn_edge_set = graph.syn_edge_set;
$$syn_id_constraint_set = cc.syn_id_constraint_set;
cc.inh_vertex_id_met_set = $$syn_vertex_id_met_set;
}/*----------------------------------------*/

local BOOL producerop_period_le_consumerop_error;
producerop_period_le_consumerop_error =

{IsProducerOp_Period_Less_Or_Equal_ConsumerOp($$.syn_edge_set,
$$.syn_id_constraint_set));

local STR producerop_period_le_consumerop_msg;
producerop_period_le_consumerop_msg =

{producerop_period_le_consumerop_error}

? "\n" # "For any edge, the Producer's Period\n" # "must exceed that of the Consumer."

: "

/*----------------------------------------*/
APPENDIX C - Attribute Rules

local BOOL sporadic_consumerop_w_trigger_error;
sporadic_consumerop_w_trigger_error =
    {Is_Sporadic_ConsumerOp_W_Trigger($$.syn_edge_set,
    $$.syn_id_constraint_set));

local STR sporadic_consumerop_w_trigger_msg;
sporadic_consumerop_w_trigger_msg =
    {sporadic_consumerop_w_trigger_error}
? 'n'
# "! For any edge, if the Consumer is an"
# " a sporadic, it must have a Trigger."
    : 'n';

/-----------------------------/

local BOOL constr_producerop_and_unconstr_consumerop_w_trigger_error;
constr_producerop_and_unconstr_consumerop_w_trigger_error =
    {Is_ConstrProducerOp_And_UnConstConsumerOp_W_Trigger($$.syn_edge_set,
    $$.syn_vertex_id_met_set,
    $$_.id_constraint_set)};

local STR constr_producerop_and_unconstr_consumerop_w_trigger_msg;
constr_producerop_and_unconstr_consumerop_w_trigger_msg =
    {constr_producerop_and_unconstr_consumerop_w_trigger_error}
? 'n'
# "! For any edge, if the Producer is constrained,"
# " an unconstrained Consumer can not have a Trigger."
    : 'n';

/-----------------------------/

local BOOL unconstr_producerop_and_constr_consumerop_w_byall_error;
unconstr_producerop_and_constr_consumerop_w_byall_error =
    {Is_UnConstProducerOp_And_ConstrConsumerOp_W_ByAll($$.syn_edge_set,
    $$.syn_vertex_id_met_set,
    $$_.id_constraint_set)};

local STR unconstr_producerop_and_constr_consumerop_w_byall_msg;
unconstr_producerop_and_constr_consumerop_w_byall_msg =
    {unconstr_producerop_and_constr_consumerop_w_byall_error}
? 'n'
# "! For any edge, if the Producer is unconstrained,"
# " a constrained Consumer triggered By_All"
# " can result in overflow."
    : 'n';

/-----------------------------/

local BOOL unconstr_producerop_and_constr_consumerop_w_bysome_error;
unconstr_producerop_and_constr_consumerop_w_bysome_error =
    {Is_UnConstProducerOp_And_ConstrConsumerOp_W_BySome($$.syn_edge_set,
    $$.syn_vertex_id_met_set,
    $$_.id_constraint_set)};

local STR unconstr_producerop_and_constr_consumerop_w_bysome_msg;
unconstr_producerop_and_constr_consumerop_w_bysome_msg =
    {unconstr_producerop_and_constr_consumerop_w_bysome_error}

? "n"
# " For any edge, if the Producer is unconstrained,"
# " a constrained Consumer triggered By_Some"
# " can result in Data loss."
    : 'n';

/-----------------------------/

local BOOL has_error;
has_error =
    (producerop_period_consumers_op_error ||
    sporadic_consumerop_w_trigger_error ||
    constr_producerop_and_unconst_consumerop_w_trigger_error ||
    unconstr_producerop_and_constr_consumerop_w_byall_error ||
    unconstr_producerop_and_constr_consumerop_w_bysome_error);

local STR error_header;
error_header = (has_error)
? "n"
# " SCHEDULING NOTICE:
    : 'n';

local STR error_trailer;
error_trailer = (has_error)
? "n"
# "* * * * * * * * * * * * * *
    : 'n';

/-----------------------------/

graph : GraphNull
    {$$_.vertex_id_set = OpIdSetNil;
    $$_.syn_vertex_id_met_set = OpIdMetSetNil;
    $$_.syn_edge_set = EdgeSetNil;
    $$_.edge_id_set = IdSetNil;
    }

Graph : Graph
    {$$_.vertex_id_set = vertex_list.vertex_id_set;
    $$_.syn_vertex_id_met_set = vertex_list.syn_vertex_id_met_set;
    $$_.syn_edge_set = edge_list.syn_edge_set;
    $$_.edge_id_set = edge_list.edge_id_set;
    }

/-----------------------------/

vertex_list : VertexListNull
    {$$_.vertex_id_set = OpIdSetNil;
    $$_.syn_vertex_id_met_set = OpIdMetSetNil;
    }

VertexListPair : VertexListPair
    {$$_.vertex_id_set = OpIdSetUnion(a_vertex.vertex_id_set,
    vertex_list.$$_.vertex_id_set);
    $$_.syn_vertex_id_met_set =
    OpIdMetSetUnion(Get_Id_Met_Set(a_vertex),
    vertex_list.$$_.syn_vertex_id_met_set);
    }

a_vertex : AVertexNull
    {$$_.vertex_id_set = OpIdSetNil;
    }

111
APPENDIX C - Attribute Rules

```plaintext
/

/
```
APPENDIX C - Attribute Rules

/*-- 'This Periodic Operator is not schedulable.' */

local BOOL unschedulable_sporadic_error;
unschedulable_sporadic_error = (!unconstrained_op_with_constraints_error
  && !invalid_time_constraint_error
  && Unschedulable_Sporadic_Op(a_constraint,
  $$\$.inh_vertex_id_met_set$$));
local STR unschedulable_sporadic_msg;
unschedulable_sporadic_msg = (unschedulable_sporadic_error)
  ? "$n"
  #-- 'This Sporadic Operator is not schedulable.'
  "$n"

/*-- */

local BOOL period_only_error;
period_only_error = (!unconstrained_op_with_constraints_error
  && unschedulable_periodic_error
  && unschedulable_sporadic_error
  && !invalid_time_constraint_error
  && PeriodOnly(a_constraint);
local STR period_only_msg;
period_only_msg = (period_only_error)
  ? "$n"
  #-- 'If left unspecified,\n" Finish Within will default to Period.'
  "$n"

/*-- */

local BOOL finish_within_only_error;
finish_within_only_error = (!unconstrained_op_with_constraints_error
  && unschedulable_periodic_error
  && unschedulable_sporadic_error
  && !invalid_time_constraint_error
  && FinishWithinOnly(a_constraint);
local STR finish_within_only_msg;
finish_within_only_msg = (finish_within_only_error)
  ? "$n"
  #-- 'If left unspecified,\n" Finish Within will default to Period.'
  "$n"

/*-- */

local BOOL mrt_only_error;
mrt_only_error = (!unconstrained_op_with_constraints_error
  && unschedulable_periodic_error
  && unschedulable_sporadic_error
  && !invalid_time_constraint_error
  && MaxResTimeOnly(a_constraint);
local STR mrt_only_msg;
mrt_only_msg = (mrt_only_error)
  ? "$n"
  #-- 'If left unspecified,\n" MaxResTime will default to MinCallPeriod + MET.'
  "$n"

/*-- */

local BOOL has_error;
has_error = (!invalid_time_constraint_error ||
  unconstrained_op_with_constraints_error ||
  unschedulable_periodic_error ||
  unschedulable_sporadic_error ||
  period_only_error ||
  finish_within_only_error ||
  mrt_only_error ||
  mcp_only_error);
local STR error_header,
error_header = (has_error)
  ? "$n"
  #-- SCHEDULING NOTICE:\n" $n"
  "$n"

/*-- */

component
/* dropped...
  Op;
  added...*/
  Nocomponent{
    $$\$.syn_vertex_id_met_set$$ = OpIdMetSetN1;
    $$\$.syn_id_constraint_set$$ = IdConstraintSetN1;
  } | Op{
    $$\$.syn_vertex_id_met_set$$ = operator_impl.syn_vertex_id_met_set;
    $$\$.syn_id_constraint_set$$ = operator_impl.syn_id_constraint_set;
    local BOOL is_root;
is_root = (id != IdNull) &&
isElement(id, (prototype.syn_root_ids));
local STR root_message;
root_message = (isElement(id, (prototype.syn_root_ids))
  ? "$n--\n" This Is A Root Operator"
  "$n"

local BOOL multiple_root_error;
multiple_root_error = (id != IdNull) &&
APPENDIX C - Attribute Rules

```plaintext
local BOOL obsolete_input_error;
obsolete_input_error = (!IsNull(obsolete_input_id));

local STR obsolete_input_message;
obsolete_input_message = (obsolete_input_error)
? "\n-- \n-- Obsolete Input Declarations \n" + ";";

local BOOL undefined_output_error;
undefined_output_error = (!IsNull(undefined_output_id));

local STR undefined_output_message;
undefined_output_message = (undefined_output_error)
? "\n-- \n-- Missing Output Declarations \n" + ";";

local BOOL obsolete_output_error;
obsolete_output_error = (!IsNull(obsolete_output_id));

local STR obsolete_output_message;
obsolete_output_message = (obsolete_output_error)
? "\n-- \n-- Obsolete Output Declarations \n" + ";";

local type Declarations inh_input_declarations;
inh_input_declarations = Build_Type_Decl(
    inh_input_id_set,
    (prototype.syn_defined_type_decl));

local type Declarations inh_output_declarations;
inh_output_declarations = Build_Type_Decl(
    inh_output_id_set,
    (prototype.syn_defined_type_decl));

local IDSet input_type_error_set;
input_type_error_set = Extract_Type_Error_Set(
    IDSetIntersect(inh_output_id_set, inh_input_id_set),
    operator_spec.inputs_types,
inh_input_declarations);

local BOOL input_type_error;
input_type_error = (!IsNull(input_type_error_set));

local STR input_type_error_message;
input_type_error_message = (input_type_error)
? "\n-- \n-- Input Type Declarations Does Not Match\n-- Stream Declarations in Parent Operators\n" + ";";

local IDSet output_type_error_set;
output_type_error_set = Extract_Type_Error_Set(
    IDSetIntersect(output_id_set, inh_output_id_set),
    operator_spec.outputs_types,
inh_output_declarations);

local BOOL output_type_error;
output_type_error = (!IsNull(output_type_error_set));

local STR output_type_error_message;
output_type_error_message = (output_type_error)
? "\n-- \n-- Output Type Declarations Does Not
```
APPENDIX C - Attribute Rules

Match -- Stream Declarations in Parent Operators
: **

local time inh_met;
inh_met = Extract_Net(

  OperatorId(

    OptionalTypeIDNull,
    1d,
    OperatorIdPairsNull),

      (prototype.syndefined_operators));

local BOOL met_error;
met_error = (multiple_root_error && Different_Time(inh_met,
operator_spec.syn_met));

local CTR met_error_message;
met_error_message = (met_error)

  "\n\n-- Met Does Not Match Vertex Time in Parent
Operator\n:

  **

local id loc_operator_id;
loc_operator_id = 1d;

local OpIdSet vertex_id_set;
vertex_id_set = operator_impl.vertex_id_set;

local IdSet edge_id_set;
edge_id_set = operator_impl.edge_id_set;

local IdSet stream_id_set;
stream_id_set = operator_impl.stream_id_set;

local OpIdSet constraint_id_set;
constraint_id_set = operator_impl.constraint_op_id_set;

local IdSet state_id_set;
state_id_set = operator_spec.state_id_set;

local IdSet obsolete_state_id;
obsolete_state_id = (Get_Impl_Form(operator_impl) == 2)
  ? IdSetDifference(state_id_set, edge_id_set)
  : IdSetNull;

  local BOOL obsolete_state_error;
obsolete_state_error = (IsNull(obsolete_state_id));

local STR obsolete_state_message;
obsolete_state_message = (obsolete_state_error)
  ? "\n\n-- Obsolete State Declarations \n"
  : "**

local IdSet input_output_state_union_set;
input_output_state_union_set =

  IdSetUnion(state_id_set,
  IdSetUnion(input_id_set, output_id_set));

local IdSet undefined_stream;
undefined_stream =

  IdSetDifference(edge_id_set,
  IdSetUnion(stream_id_set,
  input_output_state_union_set));

local IdSet obsolete_stream;
obsolete_stream =

  IdSetUnion(
  IdSetDifference(state_id_set, edge_id_set),
  IdSetIntersection(input_id_set, output_id_set));

local OpIdSet undefined_constraint;
undefined_constraint =

  IdSetDifference(
  constraint_id_set,
  vertex_id_set);

local OpIdSet obsolete_constraint;
obsolete_constraint =

  IdSetDifference(
  constraint_id_set,
  vertex_id_set);

local BOOL undefined_stream_error;
undefined_stream_error = (IsNull(undefined_stream));

local STR undefined_stream_message;
undefined_stream_message = (undefined_stream_error)
  ? "\n\n-- Missing Stream Declarations \n"
  : "**

local BOOL obsolete_stream_error;
obsolete_stream_error = (IsNull(obsolete_stream));

local STR obsolete_stream_message;
obsolete_stream_message = (obsolete_stream_error)
  ? "\n\n-- Obsolete Stream Declarations \n"
  : "**

local BOOL undefined_constraint_error;
undefined_constraint_error = (OpIdSetIsNull(undefined_constraint));

local STR undefined_constraint_message;
undefined_constraint_message = (undefined_constraint_error)
  ? "\n\n-- Missing Constraint Entries \n"
  : "**

local BOOL obsolete_constraint_error;
obsolete_constraint_error = (OpIdSetIsNull(obsolete_constraint));

local STR obsolete_constraint_message;
obsolete_constraint_message = (obsolete_constraint_error)
  ? "\n\n-- Obsolete Constraint Entries \n"
  : "**

local BOOL has_error;
has_error = (is_root ||
  multiple_op_spec_error ||
  also_defer_state_error ||
  undefined_input_error ||
  obsolete_input_error ||
  undefined_output_error ||
  obsolete_output_error ||
  input_type_error ||

115
APPENDIX C - Attribute Rules

store(default_store edge_ids_store) local IdSet local_edge_id_set;
local_edge_id_set = edge_id_set;

/*
store(default_store streams_ids_store) local IdSet local_streams_ids_set;
local_streams_ids_set = IdSetDifference(edge_id_set,
    IdSetUnion(state_id_set,
    IdSetUnion(input_id_set, output_id_set)));
*/

store(default_store constrainta_ids_store) local OpIdSet local_constraints_ids_set;
local_constraints_ids_set = constraint_op_id_set;

store(default_store stream_error_store) local BOOL local_stream_error;
local_stream_error = (undefined_stream_error || obsolete_stream_error);

store(default_store constraint_error_store) local BOOL local_constraint_error;
local_constraint_error = (undefined_constraint_error || obsolete_constraint_error);

)

| Data(

$$\text{syn_vertex_id}_\text{met.set} = \text{OpIdSet} Nil;$$
$$\text{syn_id_constraint.set} = \text{IdConstraintSet} Nil;$$
local id local_type_id;
local_type_id = id;
local o_operators syn_o_operators;
syn_o_operators = type_spec.defined_operators;
local operator_impl_list syn_operator_impl_list;
syn_operator_impl_list = type_impl.syn_operator_impl_list;
local BOOL multiple_type_spec_error;
multiple_type_spec_error = (id != IdNull) &&
IsElement(id, prototype.syn.defined_operators);
lstore STR multiple_type_spec_message;
multiple_type_spec_message = (multiple_type_spec_error)
? "\n-- \n-- Multiply Defined Types * ";

local BOOL also_defined_as_op_error;
also_defined_as_op_error = (id != IdNull) &&
(definedOp(id, prototype.syn.defined_operators) > 0);
local STR also_defined_as_op_message;
also_defined_as_op_message = (also_defined_as_op_error)
? "\n-- \n-- Id denotes both as Operator and Type* ";

local IdSet undefined_op_impl;
undefined_op_impl = (PSDLTypeImpl(type_impl)
? Extract Undefined Op_Impl In Data(type_spec.defined_operators,
    type_impl.syn.operator_impl_list)
    IdSet Nil);
local BOOL undefined_op_impl_error;
undefined_op_impl_error = (IdSetSize(defined_op_impl) > 0);
local STR undefined_op_impl_message;
undefined_op_impl_message = {undefined_op_impl_error}
? "\n-- \n-- The Following Operators Has No Operator Impl:
"
; **;

local IdSet obsolete_op_impl;
obsolete_op_impl =
Extract_Obsolete_Op_Impl_In_Data(type_impl.syn_operator_impl_list,
type_spec.defined_operators);

local BOOL obsolete_op_impl_error;
obsolete_op_impl_error = {ids.size(obsolete_op_impl) > 0};

local STR obsolete_op_impl_message;
obsolete_op_impl_message = {obsolete_op_impl_error}
? "\n-- \n-- The Following Operators Impl Are
Obsolete:
"
; **;

local BOOL has_error;
has_error = (multiple_type_spec_error ||
also_defined_as_op_error ||
undefined_op_impl_error ||
obsolete_op_impl_error
);

local STR error_header;
error_header = {has_error}
? "\n------------------------ \n-- WARNINGS, ERRORS AND
ALERTS:
"
; **;

local STR error_trailer;
error_trailer = {has_error}
? "\n------------------------
"
; **;

store(default_store t.undefined_op_impl_store) local IdSet
local undefined_op_impl_set;
local undefined_op_impl_set = undefined_op_impl;

store(default_store t.obsolete_op_impl_store) local IdSet
local obsolete_op_impl_set;
local obsolete_op_impl_set = obsolete_op_impl;

);
APPENDIX D - Auxiliary Functions

/* This first set of functions were already existing and */
/* in support of the basic SDR. */

/**/ component GetOperator(component c)
  (with(c)
    (NoComponent : NoComponent,
      htmls, ts, tlj : NOComponent,
      Op(i, os, ol) : Op(i, os, ol)
    )
  );
/**/ component GetType(component c)
  (with(c)
    (NoComponent : NoComponent,
      htmls, ts, tlj : NOComponent,
      Op(i, os, ol) : NoComponent
    )
  );

/**/ type_declarations Get_Inps(component c)
  (with(c)
    (NoComponent : TypeDeclNil,
      htmls, ts, tlj : NOComponent,
      Get_Inps(i, os, ol) : Get_Inps(i, os, ol)
    )
  );
/**/ type_declarations Get_Outps(component c)
  (with(c)
    (NoComponent : TypeDeclNil,
      htmls, ts, tlj : NOComponent,
      Get_Outps(i, os, ol) : Get_Outps(i, os, ol)
    )
  );
/**/ type_declarations Get_Opset(component c)
  (with(c)
    (NoComponent : TypeDeclNil,
      htmls, ts, tlj : NOComponent,
      Get_Opset(i, os, ol) : Get_Opset(i, os, ol)
    )
  );
/**/ type_declarations Get_Inps_Decl(component c)
  (with(c)
    (NoComponent : TypeDeclNil,
      htmls, ts, tlj : NOComponent,
      Get_Inps_Decl(i, os, ol) : Get_Inps_Decl(i, os, ol)
    )
  );
/**/ type_declarations Get_Outps_Decl(component c)
  (with(c)
    (NoComponent : TypeDeclNil,
      htmls, ts, tlj : NOComponent,
      Get_Outps_Decl(i, os, ol) : Get_Outps_Decl(i, os, ol)
    )
  );
/**/ type_declarations Get_Opset_Decl(component c)
  (with(c)
    (NoComponent : TypeDeclNil,
      htmls, ts, tlj : NOComponent,
      Get_Opset_Decl(i, os, ol) : Get_Opset_Decl(i, os, ol)
    )
  );
/**/ type_declarations Get_Inps_Decl_from_Inps(component c)
  (with(c)
    (NoComponent : TypeDeclNil,
      htmls, ts, tlj : NOComponent,
      Get_Inps_Decl_from_Inps(i, os, ol) : Get_Inps_Decl_from_Inps(i, os, ol)
    )
  );
/**/ type_declarations Get_Inps_Decl_from_Inps_list(component c)
  (with(c)
    (NoComponent : TypeDeclNil,
      htmls, ts, tlj : NOComponent,
      Get_Inps_Decl_from_Inps_list(i, os, ol) : Get_Inps_Decl_from_Inps_list(i, os, ol)
    )
  );
/**/ type_declarations Get_Outps_Decl_from_Outps(component c)
  (with(c)
    (NoComponent : TypeDeclNil,
      htmls, ts, tlj : NOComponent,
      Get_Outps_Decl_from_Outps(i, os, ol) : Get_Outps_Decl_from_Outps(i, os, ol)
    )
  );
/**/ type_declarations Get_Outps_Decl_from_Outps_list(component c)
  (with(c)
    (NoComponent : TypeDeclNil,
      htmls, ts, tlj : NOComponent,
      Get_Outps_Decl_from_Outps_list(i, os, ol) : Get_Outps_Decl_from_Outps_list(i, os, ol)
    )
  );
/**/ type_declarations Get_Opset_Decl_from_Opsets(component c)
  (with(c)
    (NoComponent : TypeDeclNil,
      htmls, ts, tlj : NOComponent,
      Get_Opset_Decl_from_Opsets(i, os, ol) : Get_Opset_Decl_from_Opsets(i, os, ol)
    )
  );
/**/ type_declarations Get_Opsets_Decl_from_Opsets_list(component c)
  (with(c)
    (NoComponent : TypeDeclNil,
      htmls, ts, tlj : NOComponent,
      Get_Opsets_Decl_from_Opsets_list(i, os, ol) : Get_Opsets_Decl_from_Opsets_list(i, os, ol)
    )
  );

119
APPENDIX D - Auxiliary Functions

/***** declarations Get_StreamDeclFromOp_ImplList (operator_ImplList oil)  
with (oil)  
{  
  (OilImplListNull: TypeDeclNil,  
  OilImplListPair(toi, tail)):  
  with (toi)  
  {  
    TopImplNull:Get_StreamDeclFromOp_ImplList (tail),  
    TopImpl(*, oil): Concat_TypeDeclList(  
      Get_StreamDeclFromOp_ImplList (oil),  
      Get_StreamDeclFromOp_ImplList (tail))  
  }  
}  
*/

/***** declarations Get_States (component c)  
with (c)  
{  
  NoComponent: TypeDeclNil,  
  Op(i, os, oil): Get_StatesDeclFromStateList (Get_StateList (os)),  
  Data(i, ts, ti):  
  with (ts)  
  {  
    TypeSpec(*, oo, *': *, *):  
      Get_StatesDeclFromO_Operators (oo)  
  }  
}  
*/

/***** declarations Get_StatesDeclFromO_Operators (o_operators oo)  
with (oo)  
{  
  OperatorNil: TypeDeclNil,  
  OperatorPair (ts, ti):  
  with (oo)  
  {  
    TopSpecNil: Get_StatesDeclFromO_Operators (tail),  
    TopSpec(*, oo): Concat_TypeDeclList(  
      Get_StatesDeclFromStateList (Get_StateList (os)),  
      Get_StatesDeclFromO_Operators (tail))  
  }  
}  
*/

/***** declarations Get_StatesDeclFromStateList (o_states_list osl)  
with (osl)  
{  
  StateListNone: TypeDeclNil,  
  StateListPair (os, tail): Concat_TypeDeclList(  
    Get_StatesDeclFromO_State (os),  
    Get_StatesDeclFromStateList (tail))  
}  
*/

/***** declarations Get_StatesDeclFromO_State (o_states os)  
with (os)  
{  
  OpStatesNone: TypeDeclNil,  
  OpStates (td, el, rt): td  
}  
*/

/***** declarations Get_VertexList (component c)  
with (c)  
{  
  NoComponent: VertexListNil,  
  Op(i, os, oil): Get_VertexList (oil),  
  Data(i, ts, ti):  
  with (ti)  
  {  
    TypeImplNull: VertexListNil,  
    AdaTypeImpl(*): VertexListNil,  
    TypeImpl(*, oil):  
      Get_VertexListFromOp_ImplList (oil)  
  }  
}  
*/

vertex_list Concat_VertexList (vertex_list v11, vertex_list v12)  
{  
  (VertexNil: v12,  
  VertexListPair (av, tail): av:: Concat_VertexList (tail, v12))  
}  
*/

edge_list GetEdges (component c)  
{  
  NoComponent: EdgeListNil,  
  Data (i, ts, ti): EdgeListNil,  
  Op (i, os, oil): Get_Edge_List (oil)  
}  
*/

edge_list Concat_EdgeList (edge_list e11, edge_list e12)  
{  
  (EdgeListNil: e12,  
  EdgeListPair (ae, tail): ae:: Concat_EdgeList (tail, e12))  
}  
*/

INT DefinedOprr (id ident, psdl_components o1)  
{  
  with (o1)  
  {  
    PsdlNil: 0,  
    PsdlPair (c, l): with (c)  
    {  
      NoComponent: DefinedOprr (ident, 1),  
      Data (i, os, oil): DefinedOprr (ident, 1),  
      Op (i, os, oil): (((ident == 1) || 0)  
        DefinedOprr (ident, 1))  
    }  
  }  
}  
*/

INT DefinedType (id ident, psdl_components o1)  
{  
  with (o1)  
  {  
    PsdlNil: 0,  
    PsdlPair (c, l): with (c)  
    {  
      NoComponent: DefinedType (ident, 1),  
      Data (i, os, oil): (((ident == 1) || 0)  
        DefinedType (ident, 1),  
      Op (i, os, oil): DefinedType (ident, 1))  
    }  
  }  
}  
*/

INT DefinedTypeOp (operator_id ident, psdl_components o1)
APPENDIX D - Auxiliary Functions

```c
{ with (ident)
  OperatorIdNull: 0,
  OperatorId(tid, oid, *):
    with (tid)
    { OptionalOperatorIdNull: 0,
      OptionalOperatorIdPrompt: 0,
      OptionalOperatorId(tod):
        with (ol)
        { PedIdNull: 0,
          PedPair(o, l):
            with [c]
            { NoComponent: DefinedTypeOp(ident, l),
              Data(l, ts, tl): ((l = ts) ||
                DefinedOpInTypeSpec(oid, ts) ==
                DefinedTypeOp(ident, l),
                Op(i, os, ol): DefinedTypeOp(ident, l)
              )
            };
        }
      }
    }
  }
};

 /*------------------------------------------------------------------------*/
INT DefinedOpInTypeSpec(id i, type_spec ts)
{ with (ts)
  TypeSpec(*, *, oo, *, *): OpIsDefinedInTypeSpec(i, oo)
};

 /*------------------------------------------------------------------------*/
INT DefinedVertex(operator_id ident, vertex_list vl)
{ with (vl)
  VertexListNull: 0,
  VertexListPair(ov, tail):
    with (av)
    { AVertexNull: DefinedVertex(ident, tail),
      AVertex(oi, *):
        with (oi)
        { OperatorIdNull: DefinedVertex(ident, tail),
          OperatorId(*, *, *):
            { EqualOpId(oi, ident) ? 1 : 0
              + DefinedVertex(ident, tail)
            }
        }
      }
    }
};

 /*------------------------------------------------------------------------*/
INT OpIsDefinedInTypeSpec(id i, o_operators oo)
{ with (oo)
  { OperatorNil: 0,
    OperatorPair(os, tail):
      with (os)
      { TopSpecNil: OpIsDefinedInTypeSpec(i, tail),
        TopSpec(tid, *): ((i = tid) ? 1 : 0) + OpIsDefinedInTypeSpec(i, tail)
      }
    }
};

 /*------------------------------------------------------------------------*/
INT OpIsDefinedInTypeImpl(id i, operator_impl_list oil)
{ with (oil)
  { OpImplListNull: 0,
    OpImplListPair(toi, tail):
      with (toi)
      { TopImplNil: OpIsDefinedInTypeImpl(i, tail),
        TopImpl(tl_id, oi):
          { (tl_id == i) ? 1 : 0
            + OpIsDefinedInTypeImpl(i, tail)
          }
      }
    }
};

/*------------------------------------------------------------------------*/
int GetDeclTypeId(decl_type_name_t)
{ with [t]
  { DTySimpleId(i): i,
    DTyUserDefined(i, *): 1,
    DTyNameNull: IDNull,
    DTyInteger: IDNull,
    DTyReal: IDNull,
    DTyBool: IDNull
  };

/*------------------------------------------------------------------------*/
INT IdIsDefined(id i, id_list idl)
{ with (idl)
  { IDPair[ix, lid]: ((i == ix) ? 1 : 0) + IdIsDefined(i, idl[x])}
};

 /*------------------------------------------------------------------------*/
INT IdIsDefinedInTypes(id ident, type_declarations td)
{ with (td)
  { TypeDeclNil: 0,
    TypeDeclPair(adl, td2): with (adl)
      { ADecl(idl, *): IdIsDefined(ident, idl) + IdIsDefinedInTypes(ident, td2)
      }
    }
};

 /*------------------------------------------------------------------------*/
type_declarations GetGenericsTypes(o_generics ogen)
{ with (ogen)
  { OGenTypesNone: TypeDeclNil,
    OGenerics(td, rt): td
  }
};

 /*------------------------------------------------------------------------*/
type_declarations GetInputsTypes(o_inputs oinp)
{ with (oinp)
  { OInputNone: TypeDeclNil,
    OInputs(td, rt): td
  }
};
```
APPENDIX D - Auxiliary Functions

```c
/*----------------------------------------*/
type_declarations GetOutputsTypes(o_outputs outp)
(with(outp)
  {OpOutputNone: TypeDeclNil,
   OpOutputs(td, rt): td
  });

/*----------------------------------------*/
type_declarations GetStatesTypes(o_states estat)
(with(estat)
  {OpStatesNone: TypeDeclNil,
   OpStates(td, el, rt): td
  });

BOOL IsOperator(component c)
{  
  with(c)
    {NoComponent: false,
     Data(i, ts, ti): false,
     Op(i, os, oi): true
    };
};

BOOL IsType(component c)
{  
  with(c)
    {NoComponent: false,
     Data(i, ts, ti): true,
     Op(i, os, oi): false
    };
};

/*----------------------------------------*/
Id GetId(component c)
{  
  with(c)
    {NoComponent: IdNull,
     Data(i, ts, ti): IdNull,
     Op(i, os, oi): with (i) {IdNull: IdNull,
                              Id(v): Id(v)}
    };
}

/*----------------------------------------*/
IdSet Extract_Root_Components(psl1_components o, vertex_list vl)
{  
  {PslNil: IdSetNil,
   PslPair(o, tail): 
     with (o)
       {NoComponent: Extract_Root_Components(tail, vl),
        Data("", ":")Extract_Root_Components(tail, vl),
        Op("", ":")
       : with (i) {IdNull: Extract_Root_Components(tail, vl),
                   Id(v): Operator_Id_Not_In_Vertex_List;
                   OperatorId{
                     OptionalTypeIdNull,
                     i,
                     OperatorIdPairsNull},
                   vl)}
  }
}

*/
IdSet Extract_Undefined_Op(vertex_list vl, psl1_components o)
{  
  {VertexListNil: IdSetNil,
   VertexListPair(av, tail):
     with (av)
       {AVerexNull: Extract_Undefined_Op(tail, o),
        AVertex(oi, ":")
       : with (oi) {OperatorIdNull: Extract_Undefined_Op(tail, o),
                     OperatorId(tld, oid, ":")
       : with (tid)
         {OptionalTypeIdNull:
           {DefinedOp[oid, o] == 0
            ? oid: Extract_Undefined_Op(tail, o)
            : Extract_Undefined_Op(tail, o)},
          OptionalTypeIdPrompt:
            {DefinedOp[oid, o] == 0
             ? oid: Extract_Undefined_Op(tail, o)
             : Extract_Undefined_Op(tail, o)},
          OptionalTypeId(
           extract_Undefined_Op(tail, o))
         : extract_Undefined_Op(tail, o)
       }}
     };
}

*/
IdSet Extract_Undefined_Type_Op(vertex_list vl, psl1_components o)
{  
  {VertexListNil: IdSetNil,
   VertexListPair(av, tail):
     with (av)
       {AVerexNull: Extract_Undefined_Type_Op(tail, o),
        AVertex(oi, ":")
       : with (oi) {OperatorIdNull: Extract_Undefined_Type_Op(tail, o),
                     OperatorId(tld, oid, ":")
       : with (tid)
         {OptionalTypeIdNull:
           Extract_Undefined_Type_Op(tail, o),
          OptionalTypeIdPrompt:
            Extract_Undefined_Type_Op(tail, o),
          OptionalTypeId(
           extract_Undefined_Type_Op(tail, o))
         : extract_Undefined_Type_Op(tail, o)
       }}
     };
}

*/
```
APPENDIX D - Auxiliary Functions

```c
/* Extract_Multiple_Op_Spec_Id(psl.components o) */
(IdSet Extract_Multiple_Op_Spec_Id(psl.components o)
 with (o) { 
    (pslNil: IdSetNil, 
    (pslPair(c, tail): 
    with (c) { 
        (NoComponent: Extract_Multiple_Op_Spec_Id(tail), 
        Data(*, *, *): Extract_Multiple_Op_Spec_Id(tail), 
        Op(i, *, *)): 
        with (i) { 
            (IdNil: Extract_Multiple_Op_Spec_Id(tail), 
            Id(v): { 
                (DefinedOp(i, tail) > 0) 
            ? IdSetUnion( 
                SingletonIdSet(i), 
                Extract_Multiple_Op_Spec_Id(tail)) 
            : Extract_Multiple_Op_Spec_Id(tail)) 
        } 
    } 
} } 
)

/* Extract_Multiple_Type_Spec_Id(psl.components o) */
(IdSet Extract_Multiple_Type_Spec_Id(psl.components o)
 with (o) { 
    (pslNil: IdSetNil, 
    (pslPair(c, tail): 
    with (c) { 
        (NoComponent: Extract_Multiple_Type_Spec_Id(tail), 
        Data(i, *, *): 
        with (i) { 
            (IdNil: Extract_Multiple_Type_Spec_Id(tail), 
            Id(v): { 
                (DefinedType(i, tail) > 0) 
            ? IdSetUnion( 
                SingletonIdSet(i), 
                Extract_Multiple_Type_Spec_Id(tail)) 
            : Extract_Multiple_Type_Spec_Id(tail)) 
        } 
    } 
} 
)

/* Extract_Multiple_TD(type_declarations td) */
(IdSet Extract_Multiple_TD(type_declarations td)
 with (td) { 
    (TypeDeclNil: IdSetNil, 
    (TypeDeclPair(ad, tail): 
    with (a) { 
        (ADeclNil: Extract_Multiple_TD(tail), 
        ADecl(idl, *): 
        IdSetUnion( 
            Extract_Multiple_Ids_In_Idl_N_TD(idl, tail), 
            Extract_Multiple_TD(tail)) 
        } 
    } 
} 
)

/* Extract_Multiple_Ids_In_Idl_N_TD(id_list idl, type_declarations td) */
(IdSet Extract_Multiple_Ids_In_Idl_N_TD(id_list idl, type_declarations td)
 with (idl) { 
    (IdNil: IdSetNil, 
    (IdPair(i, idl_tail): 
    with (i) { 
        (IdNil: Extract_Multiple_Ids_In_Idl_N_TD(idl_tail, td), 
        Id(t)): { 
            (DefinedId(i, idl_tail) + IdsDefTypes(i, td) > 0) 
        ? IdSetUnion( 
            SingletonIdSet(i), 
            Extract_Multiple_Ids_In_Idl_N_TD(idl_tail, td)) 
        : Extract_Multiple_Ids_In_Idl_N_TD(idl_tail, td) 
        } 
    } 
} 
)

/* Extract_Type_Error_Set(IdSet ids, type_declarations td1, type_declarations td2) */
(IdSet Extract_Type_Error_Set(IdSet ids, type_declarations td1, type_declarations td2)
 with(ids) { 
    (IdSetNil: IdSetNil, 
```
APPENDIX D - Auxiliary Functions

```c
IdSetPair(i, tail): {Identical_Type_Name(
    Find_Type_Name(i, td1),
    Find_Type_Name(i, td2))
? Extract_Type_Error_Set(tail, td1, td2) :
    i::Extract_Type_Error_Set(tail, td1, td2)}
}

// --------------------------------------------------------------------------
// IdSet Extract_Undefined_Op_Impl_In_Data(o_operators oo, operator_Impl_list oll)
// (with oo)
(IdSetNil:IdSetNil,
 OperatorPair(tos, tail):
 with {tos} {
     TopSpecNil: Extract_Undefined_Op_Impl_In_Data(tail, oll),
     TopSpec(tid, *):
     with {tid} {
         IdNil:Extract_Undefined_Op_Impl_In_Data(tail, oll),
         Id(*):
         ((OpIsDefinedInTypeImpl(tid, oll) == 0) ? tid::Extract_Undefined_Op_Impl_In_Data(tail, oll)
            : Extract_Undefined_Op_Impl_In_Data(tail, oll))
     }
 }
 )
)

// --------------------------------------------------------------------------
// IdSet Extract_Obsolete_Op_Impl_In_Data(o_operators oll, o_operators oo)
// (with oll)
(OpImplListNil:IdSetNil,
 OpImplListPair(toi, tail):
 with {toil} {
     TopImplNil: Extract_Obsolete_Op_Impl_In_Data(tail, oll),
     TopImpl(old, *):
     with {old} {
         IdNil:Extract_Obsolete_Op_Impl_In_Data(tail, oll),
         Id(*):
         ((OpIsDefinedInTypeSpec(old, oo) == 0) ? old::Extract_Obsolete_Op_Impl_In_Data(tail, oo)
            : Extract_Obsolete_Op_Impl_In_Data(tail, oo))
     }
 }
 )
)

// --------------------------------------------------------------------------
// IdSet Make_Input_Id_Set(o_inputs_list oll)
// (with oll)
(TopSpecNil:IdSetNil,
 TopSpec(*, oo):
 with {oo} {
     OperatorSpec(g, i, o, s, e, t, k, inf, for):
         Make_Input_Id_Set(1)
 }
)

// --------------------------------------------------------------------------
// IdSet Make_Output_Id_Set(o_outputs_list oll)
// (with oll)
(TopSpecNil:IdSetNil,
 TopSpec(*, oo):
 with {oo} {
     OperatorSpec(g, i, o, s, e, t, k, inf, for):
         Make_Output_Id_Set(1)
 })
```
APPENDIX D - Auxiliary Functions

/**
 * DTypeSimpleId(*)\1: false,
 * DTypeUserDefined(*, *)\1: false
 */

/*
 * DTypeReal:
 * with(n2)
 * \1 DTypeNameNull\1: false,
 * DTypeInteger\1: false,
 * DTypeReal\1: true,
 * DTypeBoolean\1: false,
 */

/*
 * DTypeException\1: false,
 */

/*
 * DTypeSimpleId(*)\1: false,
 * DTypeUserDefined(*, *)\1: false
 */

/*
 * DTypeBoolean:
 * with(n2)
 * \1 DTypeNameNull\1: false,
 * DTypeInteger\1: false,
 * DTypeReal\1: false,
 * DTypeBoolean\1: true,
 */

/*
 * DTypeException\1: false,
 */

/*
 * DTypeSimpleId(*)\1: false,
 * DTypeUserDefined(*, *)\1: false
 */

BOOL Identical_Type_Name(decl_type_name n1, decl_type_name n2)

{with(n1)
 * DTypeNameNull:
 * with(n2)
 * \1 DTypeNameNull\1: true,
 * DTypeInteger\1: false,
 * DTypeReal\1: false,
 * DTypeBoolean\1: false,
 */

/*
 * DTypeException\1: false,
 */

/*
 * DTypeSimpleId(*)\1: false,
 * DTypeUserDefined(*, *)\1: false
 */

DTypeInteger:

{with(n2)
 * DTypeNameNull\1: false,
 * DTypeInteger\1: true,
 * DTypeReal\1: false,
 * DTypeBoolean\1: false,
 */

/*
 * DTypeException\1: false,
 */

DTypeUserDefined(n1, *)

{with(n2)
 * DTypeNameNull\1: false,
 * DTypeInteger\1: false,
 * DTypeReal\1: false,
 * DTypeBoolean\1: false,
 */

DTypeUserDefined(n1, *),

DTypeUserDefined(n1, *)

{with(n2)
 * DTypeNameNull\1: false,
 * DTypeInteger\1: false,
 * DTypeReal\1: false,
 * DTypeBoolean\1: false,
 */

DTypeUserDefined(n1, *),

DTypeUserDefined(n1, *)

{with(n2)
 * DTypeNameNull\1: false,
 * DTypeInteger\1: false,
 * DTypeReal\1: false,
 * DTypeBoolean\1: false,
APPENDIX D - Auxiliary Functions

```c
/*
 * DTypeBoolean: false,
 */

/*
 * DTypeException: false,
 */

/*
 * DTypeInfo: false,
 * DTypeInfoDefined: (12, 24):
 */

/**
 *** This second set of functions were already existing and
 *** in for creating the interaction between the SIMGEN and the
 *** Graphic Editor and Graphic Viewer.
 *** Documented originally as file: ed.01.ssl

**********************************************************************/

list IdSet;
IdSet : exported IdSetNIl();
   | exported IdSetPair(id IdSet)
   ;

IdSet exported NullSet() { IdSetNIl ; }

BOOL exported IsNull(IdSet s) { s == IdSetNIl ; }

BOOL exported IsElement(id i, IdSet s) {
   with { s };
   IdSetNIl: false, 
   IdSetPair(head, t):{(i == head)
      ? IsElement(i, t)
      : true}
   /*
   IdSetPair(head, t):{(i < head)
      ? IsElement(i, t)
      : ((i == head)
      ? true
      : false)}
   */
}

IdSet exported SingletonIdSet(id i) {
   with { i };
   IdNIl : IdSetNIl;
   Id(*) : i :: IdSetNIl
}

IdSet exported IdSetUnion(IdSet s1, IdSet s2) {
   with { s1 }
   { IdSetNIl: s2, 
     IdSetPair(i1, t1):
     with { s2 }
     { IdSetNIl: s1, 
       IdSetPair(i2, t2)
       : i1 < i2 ? IdSetUnion(t1, s2)
       : i1 == i2 ? IdSetUnion(t1, t2)
       : i2 :: IdSetUnion(s1, t2)
     }
   }
}

IdSet exported IdSetIntersect(IdSet s1, IdSet s2) {
   with { s1 }
   { IdSetNIl: IdSetNIl, 
     IdSetPair(i1, t1):
     with { s2 }
     { IdSetNIl: IdSetNIl, 
       IdSetPair(i2, t2)
       : i1 < i2 ? IdSetIntersect(t1, s2)
       : i1 == i2 ? IdSetIntersect(t1, t2)
       : IdSetIntersect(s1, t2)
     }
   }
}

IdSet exported IdSetDifference(IdSet s1, IdSet s2) {
   with { s1 }
   { IdSetNIl: s1, 
     IdSetPair(i1, t1):
     with { s2 }
     { IdSetNIl: s1, 
       IdSetPair(i2, t2)
       : i1 < i2 ? IdSetDifference(t1, s2)
       : i1 == i2 ? IdSetDifference(t1, t2)
       : IdSetDifference(s1, t2)
     }
   }
}

id exported FirstElement(IdSet s) {
   with { s };
   { IdSetNIl: IdNIl, 
     IdSetPair(i1, t1): i1
   }
}

IdSet exported IdSetTail(IdSet s) {
   with { s };
   { IdSetNIl: IdSetNIl, 
     IdSetPair(i1, t1): t1
   }
}

INT exported IdSetSize(IdSet s) {
   with { s };
   { IdSetNIl: 0, 
     IdSetPair(i1, t1): 1 + IdSetSize(t1)
   }
}

/*********************************************************************

list OpIdSet;
OpIdSet : exported OpIdSetNIl();
   | exported OpIdSetPair(operator_id OpIdSet)
   ;

OpIdSet exported NullOpIdSet() { OpIdSetNIl ; }

BOOL exported OpIdSetIsNull(OpIdSet s) { s == OpIdSetNIl ;

126
```
APPENDIX D - Auxiliary Functions

BOOL exported OpIdIsInOpIdSet(operator_id i, OpIdSet s) {
    with [s]
    { OpIdSetNil: false,
        OpIdSetPair(old, t): [LessThanOpId(i, old)
            ? OpIdIsInOpIdSet(i, t)
            : (EqualOpId(i, old)
                ? true
                : false)]
    }
}

OpIdSet exported SingletonOpIdSet(operator_id i) {
    with [i]
    { OperatorIdNull : OpIdSetNil,
        OperatorId[* , *] : i :: OpIdSetNil
    }
}

OpIdSet exported OpIdSetUnion(OpIdSet s1, OpIdSet s2) {
    with [s]
    { OpIdSetNil: s2,
        OpIdSetPair(i1, t1):
        with [s2]
        { OpIdSetNil: s1,
            OpIdSetPair(i2, t2):
            [LessThanOpId(i1, i2)
                ? i1 :: OpIdSetUnion(t1, s2)
                : (EqualOpId(i1, i2)
                    ? i1 :: OpIdSetUnion(t1, t2)
                    : i2 :: OpIdSetUnion(s1, t2))]
        }
    }
}

OpIdSet exported OpIdSetIntersect(OpIdSet s1, OpIdSet s2) {
    with [s1]
    { OpIdSetNil: OpIdSetNil,
        OpIdSetPair(i1, t1):
        with [s2]
        { OpIdSetNil: OpIdSetNil,
            OpIdSetPair(i2, t2):
            [LessThanOpId(i1, i2)
                ? OpIdSetIntersect(t1, s2)
                : (EqualOpId(i1, i2)
                    ? i1 :: OpIdSetIntersect(t1, t2)
                    : OpIdSetIntersect(s1, t2))]
        }
    }
}

OpIdSet exported OpIdSetDifference(OpIdSet s1, OpIdSet s2) {
    with [s1]
    { OpIdSetNil: OpIdSetNil,
        OpIdSetPair(i1, t1):
        with [s2]
        { OpIdSetNil: s1,
            OpIdSetPair(i2, t2):
            [LessThanOpId(i1, i2)
                ? i1 :: OpIdSetDifference(t1, s2)
                : OpIdSetDifference(s1, t2))]
    }
}

operator_id exported OpIdSetFirstElement(OpIdSet s) {
    with [s]
    { OpIdSetNil: OperatorIdNull,
        OpIdSetPair(i1, t1): i1
    }
}

OpIdSet exported OpIdSetTail(OpIdSet s) {
    with [s]
    { OpIdSetNil: OpIdSetNil,
        OpIdSetPair(i1, t1): t1
    }
}

INT exported OpIdSetSize(OpIdSet s) {
    with [s]
    { OpIdSetNil: 0,
        OpIdSetPair(i1, t1): 1 + OpIdSetSize(t1)
    }
}

BOOL exported EqualOpId(operator_id i1, operator_id i2) {
    with [i1]
    { OperatorIdNull:
        with [i2]
        { OperatorIdNull: true,
            OperatorId[* , *]: false
        }
    }
}

OperatorId[tid_1, id_1, oip_1]:
    with [i2]
    { OperatorIdNull: false,
        OperatorId[tid_2, id_2, oip_2]:
        [EqualTypeId(tid_1, tid_2) &&
            (id_1 == id_2) &&
            EqualOperatorIdPair(oip_1, oip_2)]
    }
}

BOOL exported EqualTypeId(optional_type_id tid_1, optional_type_id tid_2) {
    with [tid_1]
    { OptionalTypeIdNull:
        with [tid_2]
        { OptionalTypeIdNull: true,
            OptionalTypeIdPrompt: true,
            OptionalTypeId[*]: false
        }
    }
}

OptionalTypeIdPrompt:
    with [tid_2]
    { OptionalTypeIdNull: true,
        OptionalTypeIdPrompt: true,
        OptionalTypeId[*]: false
    }
}
APPENDIX D - Auxiliary Functions

```
},

OptionalTypeId(id_1);
with (tid_2) {
  OptionalTypeIdNull:true,
  OptionalTypeIdPrompt:true,
  OptionalTypeId(id_2) == id_1
}
}

BOOL EqualOperatorIdPairs(operator_id_pairs oip_1, operator_id_pairs oip_2) {
  with (oip_1) {
    OperatorIdPairsNull:
      with (oip_2) {
        OperatorIdPairsNull:true,
        OperatorIdPairsPrompt:true,
        OperatorIdPairs('*,*'):false
      }
  }

  OperatorIdPairsPrompt:
    with (oip_2) {
      OperatorIdPairsNull:true,
      OperatorIdPairsPrompt:true,
      OperatorIdPairs('*,*'):false
    }

  OperatorIdPairs(all_11, all_12):
    with (oip_2) {
      OperatorIdPairsNull:false,
      OperatorIdPairsPrompt:false,
      OperatorIdPairs(all_21, all_22):
        (EqualAIDLIST(all_11, all_21)
         && EqualAIDLIST(all_12, all_22))
    }
  }
}

BOOL EqualAIDLIST(alone_id_list all_1, alone_id_list all_2) {
  with (all_1) {
    AIDNULL:
      with (all_2) {
        AIDNULL: true,
        AIDPair('*,*'):false
      }
  }

  AIDPair(id_1, tail_1):
    with (all_2) {
      AIDNULL: false,
      AIDPair(id_2, tail_2):
        (id_1 == id_2) &&
        EqualAIDLIST(tail_1, tail_2))
    }
  }
}

BOOL LessThanOperatorIdPairs(operator_id_pairs oip_1, operator_id_pairs oip_2) {
  with (oip_1) {
    OperatorIdPairsNull:
      with (oip_2) {
        OperatorIdPairsNull:false,
        OperatorIdPairsPrompt:false,
        OperatorIdPairs('*,*'):true
      }
  }

  OperatorIdPairsPrompt:
    with (oip_2) {
      OperatorIdPairsNull:false,
      OperatorIdPairsPrompt:false,
      OperatorIdPairs('*,*'):true
    }

  OperatorIdPairs(all_11, all_12):
    with (oip_2) {
      OperatorIdPairsNull:false,
      OperatorIdPairs('*,*'):true
    }
}

BOOL LessThanTypeId(optional_type_id tid_1, optional_type_id tid_2) {
  with (tid_1) {
    OptionalTypeIdNull:
      with (tid_2) {
        OptionalTypeIdNull:false,
        OptionalTypeIdPrompt:false,
        OptionalTypeId('*'):true
      }
  }

  OptionalTypeIdPrompt:
    with (tid_2) {
      OptionalTypeIdNull:false,
      OptionalTypeIdPrompt:false,
      OptionalTypeId('*'):true
    }

  OptionalTypeId(id_1):
    with (tid_2) {
      OptionalTypeIdNull:false,
      OptionalTypeIdPrompt:false,
      OptionalTypeId(id_2) < id_1
    }
}
```

128
OperatorIdPairsPrompt: false,
OperatorIdPairs(all_11, all_22):
  (LessThanAIDLST(all_11, all_22) ||
   EqualAIDLST(all_11, all_22) &&
   LessThanAIDLST(all_12, all_22)
  )
}

BOOL LessThanAIDLST(alone_id_list all_1, alone_id_list all_2)
{ with(all_1)
  ( allNil:
    with(all_2)
    ( allNil: false,
      AIDPair("*"):true
    ),
    AIDPair(id_1, tail_1):
      with(all_2)
      ( allNil: false,
        AIDPair(id_2, tail_2):
          (id_1 < id_2) ||
           (id_1 == id_2) &&
           LessThanAIDLST(tail_1, tail_2)
        )
      )
  )
}

/*------------------------------------------*/
list state_list:
state_list:
  exported StateListNil()
  | exported StateListPair(a_state_pair state_list)

/* (id id) = (operator_id type_id) */
a_state_pair:
  exported AStatePair(id id);
list coor_list:
coor_list:
  exported CoorListNil()
  | exported CoorListPair(a_coordinate coor_list)

a_coordinate:
  exported XYValues(INT INT);

/*------------------------------------------*/
list id_to_met_list:
id_to_met_list:
  exported IDToMetListNil()
  | exported IDToMetListPair(a_pair id_to_met_list)

a_pair:
  NumberNamePair(id INT);

/*------------------------------------------*/

// VtxNetListPair(vtx_met_pair vtx_met_list):

vtx_met_pair :
  VtxNetPair(id INT);

  | VtxNetPair(id INT):
    with(i) (IdNull: i,
      Id(*): 0)
  );

  | VtxNetListNil(vtx_met_list)
    with[i] (VtxNetListNil: i,
      VtxNetListPair(*, *): 0)
  );

  | VtxNetListNull(a_vertex v)
    with[v] (VtxNetListNull: v,
      AVertex(*, *): 0)
  );

  | Exported Get_Op_Name(component p)
    with(p) {
      Op(1, *): Get_Id(i),
      NoComponent: **,
      Data(*, *, *): **
    }
  );

  | Exported Get_Operator_Spec(component p)
    with(p) {
      Op(*, os, *): os
    }
  );

  | Exported Get_Operator_Impl(component p)
    with(p) {
      Op(*, *), oi, 0
    }

  | Exported Get_Impl_Form(component_impl oi)
    with[oi] (OpImplNull: 0,
      AdaOpImpl(*): 0,
      OperatorImpl(g, *, *):
        with(g) (GraphNull: 1,
          Graph(*, *): 2)
  );
APPENDIX D - Auxiliary Functions

BOOL exported Op_Impl_Is_Null(operator_impl oi) { with (oi) { (OpImplNull): true, AdaOpImpl(*): false, OperatorImpl(*, *, *): false } }

BOOL exported IsTypeDeclNil(type_declarations td) { with (td) { (TypeDeclNil): true, TypeDeclPair(*, *): false } }

BOOL exported IsADeclNil(a_decl ad) { with (ad) { (ADeclNil): true, ADecl(*, *): false } }

BOOL exported IsIdListNil(id_list idl) { with (idl) { (IdNil): true, IdPair(*, *): false } }

id exported GetId_From_Id_List(id_list idl) { with (idl) { (IdNil): IdNull, IdPair(id, *): id } }

constraints exported Get_Constraints_From_Op_Impl(operator_impl oi) { with(oi) { (OpImplNull): ConstraintsNil, AdaOpImpl(*): ConstraintsNil, OperatorImpl(*, *, C): with(c) { (CcNull): ConstraintsNil, Cc(c, *, *): c } } }

optional_streams exported Make_StreamNull() { StreamNull };

constraints exported Make_ConstraintsNull() { ConstraintsNil };

/* the following routine is commented out because it is not longer being used */
a_constraint exported Make_AConstraint_From_Op_Impl(operator_id i) { AConstraint(i, OptionalTriggerNull, OptPeriodNull, OptFinishWithinNull, OptMrgNull, OptMrtNull, OutputGuardsNil, ExceptionOpsNull, TimerOperationsNil) }

/* */
a_constraint exported Make_AConstraintNull() { AConstraintNull }

/* */
id exported Make_Id_From_SSLstring(STR x) { Id(x) };

id exported Make_IdNull() { IdNull }

id_list exported Make_Id_List_Nil() { IdNil }

id_list exported Make_Id_List(id i, id_list idl) { IdPair(i, idl) }

a_decl exported Make_A_Decl_Nil() { ADeclNil }

/* */
APPENDIX D - Auxiliary Functions

```c
a_decl exported Make_A_DeclFromArray(id i)
{
    ADecl(IdPair(i, IdNil), DTypeNameNull);
}

a_decl exported Make_A_DeclPair(id_list id, decl_type_name dt)
{
    ADecl(idl, dt);
}

type_declarations exported Get_Streams_DeclFromArray_Op_Impl(operator_impl oi)
{
    with(oi)
    {
        (OpImplNull: true, AdOpImpl(*): false, OperatorImpl(*, *): false); with(oi)
    }
}

BOOL exported Op_Impl_Has_Null_Streams(operator_impl oi)
{
    with(oi)
    {
        (OpImplNull: false, AdOpImpl(*) : false, OperatorImpl(*, *): false); with(c)
    }
}

BOOL exported IsAConstraintNull(a_constraint ac)
{
    with (ac)
    {
        (AConstraintNull: true, AConstraint(*, *, *, *, *, *, *): false)
    }
}

BOOL exported IsConstraintsNull(constraints c)
{
    with (c)
    {
        (ConstraintsNull: true, ConstraintsPair(*, *): false)
    }
}

BOOL exported Valid_Net(time t)
{
    with(t)
    {
        (TimeNull: false, Time(*, *): true)
    }
}

time exported Get_Net_From_Op_Spec(operator_spec p)
{
    with(p)
    {
        OperatorSpec(*, *, *, *, t, *, *, *);
    }
    OpTimingInfo: TimeNull,
```
APPENDIX D - Auxiliary Functions

BOOL Id_Not_In_TypeImpl_Vertices(id i, type_impl ti)
{  
  with (ti)
  {   
    TypeImplNull: true,
    AdaTypeImpl(*): true,
    TypeImpl(*, oil): Id_Not_In_TypeImplList(i, oil)
  }
}

BOOL Id_Not_In_TypeImplList(id i, operator_impl_list oil)
{  
  with (oil)
  {   
    OpImplListNull: true,
    OpImplListPair{oi, tail}:
    with (oi)
    {   
      TopImplNull: true,
      TopImpl(*, o_impl):
      {   
        OperatorId_Not_In_Vertex_List{
          OperatorId(Null, i),
          OperatorIdPairNull),
          Get_Vertex_List(o_impl))
          ? Id_Not_In_TypeImplList(i, tail)
          : false
      }
    }
  }
}

STR exported Get_Vertex_Type_Id_Name(a_vertex a)
{  
  with(a)
  {   
    AVertexNull: "",
    AVertex oid, *):
    with(oid)
    {   
      OperatorIdNull: "",
      OperatorId(x1, *, *):
      {   
        with (ti)
        {   
          OptionalTypeImplNull: "",
          OptionalTypeImplPrompt: "",
          OptionalTypeId(i): Get_Id(i)
        }
      }
    }
  }
}

STR exported Get_Vertex_Operator_Id_Name(a_vertex a)
{  
  with(a)
  {   
    AVertexNull: "",
    AVertex oid, *):
    with(oid)
    {   
      OperatorIdNull: "",
      OperatorId(*, i, *): Get_Id(i)
    }
  }
}

operator_id_pairs exported Get_Vertex_OpIdPairs(a_vertex a)
{  
  with(a)
  {   
    AVertexNull: OperatorIdPairsNull,
    AVertex oid, *):
  }
}
APPENDIX D - Auxiliary Functions

```c
LatencyTimeNull; TimeNull,
LatencyTimePrompt; TimeNull,
LatencyTime(t): t
}

/*----------------------------------------------*/
time exported Get_Edge_Time(an_edge_time)
{
  with(a)
  {AnEdgeNull; TimeNull,
   AnEdge(*, lt, *); Get_Time_From_Latency(lt)
  }
}

/*****************************************************************************/
INT exported Get_Time_Value(time t)
{
  with(t)
  {TimeNull: 0,
   Time(tv, *);
   with(tv)
   {IntegerNull : 0,
    IntegerVal(digits) : STRtoINT(digits)
    }
  }
}

/*****************************************************************************/
INT exported Convert_Time_To_Integer(time t)
{
  with(t)
  {TimeNull: 0,
   Time(tv, tu);
   with(tv)
   {IntegerNull: 0,
    IntegerVal(digits):
    with(tu)
    {UnitNull: STRtoINT(digits),
     UnitMICROSECONDS: (STRtoINT(digits) / 100),
     UnitMS: STRtoINT(digits),
     UnitSEC: (STRtoINT(digits) * 1000),
     UnitMIN: (STRtoINT(digits) * 60000),
     UnitHOURS: (STRtoINT(digits) * 3600000)
    }
   }
  }
}

/*****************************************************************************/
INT exported Get_Time_Unit(time t)
{
  with(t)
  {TimeNull: 0,
   Time(*, tu):
   with(tu)
   {UnitNull: 0,
    UnitMICROSECONDS: 1,
    UnitMS: 2,
    UnitSEC: 3,
    UnitMIN: 4,
    UnitHOURS: 5
   }
  }
}

vertex_list exported Get_Vertex_List_From.Operator_Impl_List(operator_impl_list p)
{
  with (p)
  {OpImplListNull; VertexListNull,
   OpImplListPair{to, tail}:
   with {to}
   {TopImplNull;
    Get_Vertex_List_From.Operator_Impl_List{to},
    TopImpl(*, o1);
    Concat_Vertex_List{
     Get_Vertex_List(o1),
     Get_Vertex_List{to},
    }
   }
  }
}

vertex_list exported Get_Vertex_List(operator_impl p)
{
  with (p)
  {OpImplNull; VertexListNull,
   AdsOpImpl{g, *}; VertexListNull,
   OperatorImpl{g, *};
   with {g}
   {GraphNull; VertexListNull,
    Graph{vl, *}; vl
   }
  }
}

vertex_list exported Rest_Of_Vertex_List(vertex_list p)
{
  with (p)
  {VertexListNull; VertexListNull,
   VertexListPair{v, *}; v
  }
}

a_vertex exported Get_Vertex(vertex_list p)
{
  with (p)
  {VertexListNull; AVertexNull,
   VertexListPair{v, *}; v
  }
}

dge_list exported Get_Edge_List(operator_impl p)
{
  with (p)
  {OpImplNull; EdgeListNull,
   AdsOpImpl{g, *}; EdgeListNull,
   OperatorImpl{g, *};
   with {g}
   {GraphNull; EdgeListNull,
    Graph{*, el}; el
   }
  }
}

dge_list exported Rest_Of_Edge_List(edge_list p)
{
  with (p)
  
```
APPENDIX D - Auxiliary Functions

/*-----------------------------*/
| EdgeListNil: EdgeListNil,
| EdgeListPair(*, el): el
|
|); /*-----------------------------*/

an_edge exported Get_Edge(edge_list p)
{| with (p)
|   EdgeListNil: AnEdgeNull,
|   EdgeListPair(e, *): e
|); /*-----------------------------*/

o_inputs_list exported Get_Inputs_List(operator_spec p)
{| with (p)
|   OperatorSpec(*, oil, *, *, *, *, *): oil
|); /*-----------------------------*/

o_outputs_list exported Get_Outputs_List(operator_spec p)
{| with (p)
|   OperatorSpec(*, *, ool, *, *, *, *): ool
|); /*-----------------------------*/

o_states_list exported Get_State_List(operator_spec p)
{| with (p)
|   OperatorSpec(*, *, *, oel, *, *, *, *): oel
|); /*-----------------------------*/

o_states_list exported Rest_Of_State_List(o_states_list p)
{| with (p)
|   StatesListNone: StatesListNone,
|   StatesListPair(*, oel): oel
|); /*-----------------------------*/

o_states exported Get_State(o_states_list p)
{| with (p)
|   StatesListNone: OpStatesNone,
|   StatesListPair(os, *): os
|); /*-----------------------------*/

STR exported Get_State_Name(o_states s)
{| with (s)
|   OpStatesNone: "", OpStates(td, *): *;
|   with (td)
|     TypeDeclNil: "", TypeDeclPair(ad, *): ad
|     with (ad)
|       ADeclNil: "", ADeclPair(l, *): l
|   with (l)
|     IdNil: "", IdPair(l, *): Get_Id(l)
|   with (Id)
|     IdNull: "", Id(x): x
|); /*-----------------------------*/

STK exported Get_Id(id i)
{| with (i)
|   IdNull: "", Id(x): x
|); /*-----------------------------*/

STK exported Get_Decl_Type_Name(dec_type_name dtn)
{| with (dtn)
|   DTypeNull: "", DTypeInteger: "Integer",
|   DTypeReal: "Real",
|   DTypeBoolean: "Boolean",
|   DTypeException: "Exception",
|   DTypeSimpleId(i): Get_Id(i),
|   DTypeUserDefined(*): Get_Id(i)
|); /*-----------------------------*/

graph exported Return_Graph(graph G)
{| with (G)
|   GraphNull: GraphNull, Graph(VertexListNull, *): GraphNull,
|   Graph(*, *): G
|); /*-----------------------------*/

graph exported Get_Graph_From_Op_Impl(operator_impl p)
{| with (p)
|   (OpImplNull: GraphNull, AdaImpl(*)): GraphNull,
|   OperatorImpl(g, *): g
|); /*-----------------------------*/

BOOL exported Empty_Graph(operator_impl p)
{| with (p)
|   OpImplNull: true, AdaImpl("*"): true,
|   OperatorImpl(g, *): *
|   with (g)
|     GraphNull: true,
|     Graph(VL, *): *
|     with (VL)
|       VertexListNull: true,
|       VertexListPair(*, *): false
|   )
|) /*-----------------------------*/

135
APPENDIX D - Auxiliary Functions

```c
/* type declarations exported Get_Stream_decl_From_Op_impl(operator_impl p) */

with (p)
  (OpImplNull: TypeDeclNil,
   AdaOpImpl(*): TypeDeclNil,
   OperatorImpl(*, 4, *):
     with (d)
       (/* DeclarationsNull: TypeDeclNil, */
         Declarations(os, *):
           with (os)
             (StreamsNull: TypeDeclNil,
              StreamsPrompt: TypeDeclNil,
              Streams(td): td)
       )
  )

/* the following routine is commented out because it is no longer being used */
operator Id exported Get_Operator_Id_From_AConstraint(e: constraint ac)
  (with (ac)
    (AConstraintNull: IdNull,
     AConstraint(i, *, *, *, *, *, *): i)
  )

/* declarations exported Make_Null_Dependencies() */
  (DeclarationsNull
  )

optional_streams exported Build_Streams(IdSet ids, type_declarations td)
  (Streams(Build_Type_Decl(ids, td))
  )

/* type declarations exported Build_Type_Decl(IdSet ids, type_declarations td) */
  (with(ids)
    IdSetNull:
    IdSetPair(i, tail): ADecl[i, [id1], Find_Type_Name(i, td)]
    Build_Type_Decl(tail, td)
  )

/* constraints exported Build_Constraints(OpIdSet ids, constraints cs) */
  (with(ids)
    O片刻Null:
    O片刻Pair(i, tail): Build_A_Constraint(i, cs)
    Build_Constraints(tail, cs)
  )

/* constraint Build_A_Constraint(operator_id i, constraints cs) */
  (with(cs)
    (ConstraintsNull: AConstraint(i,
       OptionalTriggerNull,
       Opt_PeriodNull,
       OptFinishWithinNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
       OptNotNull,
APPENDIX D - Auxiliary Functions

OptNull, OutputGuard, ExceptionOpsNull, TimerOperationsNull), ConstraintsPair(ac, tail):
  with(ac)
  { AConstraintNull: Build_A_Constraint(i, tail),
    AConstraint(a_id, "", ",", ",", ",", ",")
    (EqualOpId(i, a_id)
     ? ac
     : Build_A_Constraint(i, tail))
  }
}

/* component exported Find_Component(id i, pedl_components pc)
   */
(component exported Find_Component(id i, pedl_components pc)
   with(pc)
   { PediNil: NoComponent,
     PediPair((c, tail):
       with(c)
       { NoComponent: Find_Component(i, tail),
         Date(c_id, ",", ",");
           (c_id == i)?: Find_Component(i, tail),
         Op0_id, ",", ",");
           (Op0_id == i)?: Find_Component(i, tail)
       }
   }
}

/* component exported Find_TopicImpl_in_operator_impl_list(id i, operator_impl_list ol)
   */
(t_op_impl exported Find_TopicImpl_in_operator_impl_list(id i, operator_impl_list ol)
   with(ol)
   { (OpImplListNull: TOPImplNull,
     OpImplListPair(t, tail):
       with(t)
       { TOPImplNull: Find_TopicImpl_in_operator_impl_list(i, tail),
         TOPImpl(t_id, ",");
           (t_id == i)?: Find_TopicImpl_in_operator_impl_list(i, tail)
       }
   }
}

/* a_constraint exported Find_A_Constraint(operator_id i, constraints ct)
   */
a_constraint exported Find_A_Constraint(operator_id i, constraints ct)
   with(ct)
   { (ConstraintsNull: AConstraintNull),
     ConstraintsPair(ac, tail):
     with(ac)
     { AConstraintNull: Find_A_Constraint(i, tail),
       AConstraint(a_id, ",", ",", ",", ",", ",", ",", ",", ",")
       (EqualOpId(a_id, i)
        ? ac
        : Find_A_Constraint(i, tail))
     }
   }

/* o_inputs_list exported Make_Input_list(type_declarations td)
   */
o_inputs_list exported Make_Input_list(type_declarations td)
   with(td)
   { (TypeDeclNil: InputsListNone),
     TypeDeclPair(*, *): InputsListPair(
       OpInputs(td, RegsTraceNone),
       InputsListNone)
   }

/* the following code separates the id_lists into individual input_lists */
/* not use in current version of pedl_editor */
/* TypeDeclPair(ad, tail): InputsListPair(
   OpInputs(137)
APPENDIX D - Auxiliary Functions

```c
TypeDeclPair(
    ad,
    TypeDeclNil),
RegmStsTraceNone),
Make_InitCalls(tail)
*/
}

/*
---
a_outputs_list exported Make_Outputs_List(type_declarations td)
(with(td)
    (TypeDeclNil:OutputsListNone,
     TypeDeclPair(ad, tail):OutputsListPair(
         CplOuts:td, RegmStsTraceNone),
         OutputsListNone)
/*

/* the following code separates the id_lists into individual output_lists */
/* not use in current version of pedl_editor */
/*
TypeDeclPair(ad, tail):OutputsListPair(
    CplOuts:
    TypeDeclPair(
        ad,
        TypeDeclNil),
        RegmStsTraceNone),
Make_Outputs_List(tail))
*/
}

--
operator_spec exported Replace_Input_Output_Id{operator_spec os,
    a_outputs_list new_l1,
    a_outputs_list new_l2,
    a_timing_info new_met}
(with(os)
    (OperatorSpec(gl, l1, ol, sl, ti, k, id, fd),
     OperatorSpec(gl, new_l1, new_ol, sl, ti, new_met, k, id, fd))
)

--
operator_impl exported Replace_Stream_Constraint_List(operator_impl ol,
    optional_streams new_stream_list,
    constraints new_constraint_list)
(with(ol)
    (OperatorImpl(g, d, c),
     OperatorImpl(g, Make_Declarations(d, new_stream_list),
     Make_Cc(c, new_constraint_list)))
)

--
*/
type_declarations exported Concat_Type_Decl_List(type_declarations td1,
type_declarations td2)
(with(td1)
    (TypeDeclNil:td1,
     TypeDeclPair(ad, tail):ad::Concat_Type_Decl_List(tail, td2)
    )
)
```

}
APPENDIX D - Auxiliary Functions

DECLAREATIONS
with (new_s)
{ StreamsNull: DeclarationsNull,
  StreamPrompt: DeclarationsNull,
  Streams(*): Declarations(new_s, TimersNull) }
,
Declarations(*, ot):
with (new_s)
{ StreamsNull:
  with (ot)
  { TimersNull: DeclarationsNull,
    TimersPrompt: DeclarationsNull,
    Timers(*): Declarations(new_s, ot) }
  ,
  StreamPrompt: Declarations(new_s, ot),
  Streams(*): Declarations(new_s, ot) }
/*
Declarations(*, ot): Declarations(new_s, ot)
*/

/*
--------------------------------------------------------------------------*/
\textbf{a\_}\textbf{timing\_}\textbf{info} exported \textbf{Make\_New\_O\_Timing\_Info}(time new\_time, a\_timing\_info old\_info)
{
  with (new_time)
  { TimeNull: OpTimingInfoNone,
    Time(*, *):
      with (old_info)
      { OpTimingInfoNone: OpTimingInfo(new_time, RegemTraceNone),
        OpTimingInfoPrompt: OpTimingInfo(new_time, RegemTraceNone),
        OpTimingInfo(*, req): OpTimingInfo(new_time, req) }
  }
}

/*
--------------------------------------------------------------------------*/
pidl\_components exported \textbf{Concat\_Pidl\_Components}(pidl\_components \textit{ol}, pidl\_components \textit{o2})
{
  with (ol)
  { PidlNil1,02, PidlPair(c, tail): Concat\_Pidl\_Components(tail, c2) }
}

/*
--------------------------------------------------------------------------*/
pidl\_components exported \textbf{Make\_New\_Ops}(IdSet \textit{op\_set})
{
  with (op\_set)
  { IdSetNil: PidlNil,
    IdSetPair(i, tail): Op[i,
    OperatorSpec:
      GenericsListNone,
      InputListNone,
      OutputListNone,
      StatesListNone,
      ExclListNone,
      OpTimingInfoNone,
      KeyWordsNone,
      InformalDescsNull,
      FormalDescsNone),
  }
}

\textbf{OpImplNil})
:: Make\_New\_Ops(tail)
}


// component exported \textbf{Make\_New\_Op\_Component}(id \textit{i}, pidl\_components \textit{o}, type\_declarations \textit{td})
{ with (i)
  { IdNil: NoComponent,
    Id(*):
      Op[i,
      OperatorSpec{
        GenericsListNone,
        InputListNone,
        OutputListNone,
        Build\_InputList(i, op, td),
        Build\_OutputList(i, o, td),
        StatesListNone,
        ExclListNone,
        OpTimingInfoNone,
        Build\_Net{
          OperatorId{
            OptionalType(iNull, i, 0),
            OperatorIdPairsNull, 0, KeyWordsNone,
            InformalDescsNull, FormalDescsNone),
            OpImplNil}
        }
      } }
}

/*
--------------------------------------------------------------------------*/
pidl\_components exported \textbf{Make\_Pidl\_Pair\_With\_Single\_Component}(component \textit{c})
{
  PidlPair(c, PidlNil)
}

/*
--------------------------------------------------------------------------*/
pidl\_components exported \textbf{Make\_Pidl\_Nil}()
{ PidlNil
}

/*
--------------------------------------------------------------------------*/
operator\_impl\_list exported \textbf{Make\_New\_Op\_Impl\_List}(IdSet \textit{op\_set})
{
  with (op\_set)
  { IdSetNil: OpImplListNull,
    IdSetPair(i, tail): OpImplListPair(
  }
}
APPENDIX D - Auxiliary Functions

```c

operator_impl_list exported Concat_Op_Impl_List(
    operator_impl_list oil_1, operator_impl_list oil_2)
{
    if [oil_1]
    [OpImplListNull:oil_2,
    OpImplListPair(toi, tail):OpImplListPair(
        toi,
        Concat_Op_Impl_List(tail, oil_2))
}

/*----------------------------------------------*/
type_impl exported Make_TypeImpl(operator_impl_list oil)
{
    TypeImpl(TableNameNull, oil)
}

/*----------------------------------------------*/
IdSet Extract_From_Edge_Ids(id i, edge_list el)
{
    EdgeListNil: IdSetNil,
    EdgeListPair(ae, tail):
    with [ae]
    [AnEdgeNull:
        Extract_From_Edge_Ids(i, tail),
        AnEdge(ei, lt, fv, tv):
        with [fv]
        [FVertexIdNull:
            Extract_From_Edge_Ids(i, tail),
            FVertexId(*, f_id, *):
            [i == f_id]
            ? IdSetUnion{
                SingletonIdSet{ei},
                Extract_From_Edge_Ids(i, tail)}
            : Extract_From_Edge_Ids(i, tail)]
        : Extract_From_Edge_Ids(i, tail)]
}

/*----------------------------------------------*/
IdSet exported Extract_Output_Id_Set(id i, psd1_components o)
{
    with [o]
    [PsdlNil:IdSetNil,
    PsdlPair(c, tail):
    with [c]
        [NoComponent:
            Extract_Output_Id_Set(i, tail),
            Data('*, *', ti):((Id_Not_In_TypeImpl_Vertices(i, ti))
            ? Extract_Output_Id_Set(i, tail)
            : Extract_Output_Id_Set_From_TypeImpl(i, ti))
        : Op('*, *_imp):
            [(Operator_Id_Not_In_Vertex_List{
                OperatorId{
                    OptionalTypeIdNull,
                    i,
                    OperatorIdPairNull,
                    Get_Vertex_List(o_imp))
            )
            : Get_Edge_List(o_imp)]
        : Get_Edge_List(o_imp)]
}

/*----------------------------------------------*/
IdSet exported Extract_From_Edge_Ids(id i, edge_list el)
{
    EdgeListNil: IdSetNil,
    EdgeListPair(ae, tail):
    with [ae]
    [AnEdgeNull:
        Extract_To_Edge_Ids(i, tail),
        AnEdge(ei, lt, fv, tv):
        with [tv]
        [TVertexIdNull:
            Extract_To_Edge_Ids(i, tail),
            TVertexId(*, t_id, *):
            [i == t_id]
            ? IdSetUnion{
                SingletonIdSet{ei},
                Extract_To_Edge_Ids(i, tail)}
            : Extract_To_Edge_Ids(i, tail)]
        : Extract_To_Edge_Ids(i, tail)]
}

/*----------------------------------------------*/
IdSet exported Extract_Output_Id_Set_From_TypeImpl(id i, type_impl ti)
{
    with [ti]
    [TypeImplNull:IdSetNil,
    AddTypeImpl('*):IdSetNil,
    TypeImpl(*, oil):Extract_Output_Id_Set_From_OpImplList(i, oil)]
}

/*----------------------------------------------*/
IdSet exported Extract_Input_Id_Set(id i, operator_impl_list oil)
```
APPENDIX D - Auxiliary Functions

```c
(with (oil)
  (OpImplListNull;IdSetNil,
   OpImplListPair(toi, tail);
   with (toi)
     (TopImplNull;IdSetNil,
      TopImpl (*, o_impl);
       (Operator_Id_Not_In_Vertex_List(
         OperatorId(
           OptionalTypeIdNull,
           i,
           OperatorIdPairsNull),
         Get_Vertex_List(o_impl))
      )
      ; Extract_Input_Id_Set_From_OpImplList(i, tail)
      ; Extract_To_Edge_Ids(i, Get_Edge_List(o_impl))
     )
  )
)

/* exported Extract_Output_Id_Set_From_TypeImpl(id i, type_impl ti) */
(idSet exported Extract_Output_Id_Set_From_TypeImpl(id i, type_impl ti))
   (with (ti)
     (TypeImplNull;IdSetNil,
      AdaTypeImpl(*, IdSetNil,
      TypeImpl (*, oil): Extract_Output_Id_Set_From_OpImplList(i, oil)
     )
   )
)

/* exported Extract_Output_Id_Set_From_OpImplList(id i, operator_impl_list oil) */
(idSet exported Extract_Output_Id_Set_From_OpImplList(id i, operator_impl_list oil))
   (with (oil)
     (OpImplListNull;IdSetNil,
      OpImplListPair(toi, tail);
      with (toi)
        (TopImplNull;IdSetNil,
         TopImpl (*, o_impl);
          (Operator_Id_Not_In_Vertex_List(
            OperatorId(
              OptionalTypeIdNull,
              i,
              OperatorIdPairsNull),
            Get_Vertex_List(o_impl))
          )
          ; Extract_Output_Id_Set_From_OpImplList(i, tail)
          ; Extract_From_Edge_Ids(i, Get_Edge_List(o_impl))
         )
      )
   )
)

/* exported Make_InputsListPair(type_declarations td) */
(InputsListPair(
   OpInputs(td, RegsetsTraceNone),
   InputsListNone)
)

/* exported Make_InputsListNone() */
(InputsListNone)

/* exported Make_OpTimingInfo.From_Het(time t) */
(OpTimingInfo(t, RegsetsTraceNone)
)

/* exported Make_OpTimingInfoNone() */
(OpTimingInfoNone)

/* exported Make_Op_From_SSLString(STR name) */
(Op(Id(name),
   OperatorSpec(
     GenericsListNone,
     InputsListNone,
     OutputsListNone,
     StatesListNone,
     ExcListNone,
     OpTimingInfoNone,
     KeywordsNone,
     InformalDescsNull,
     FormalDescsNone),
   OperatorImpl(
     GraphNull,
     * DeclarationsNull,
     * Declarations(StreamsNull, TimersNull),
     CCNull)
   )
)

/* the following routine is commented out because it is no longer being used */
/* NCOL exported Return_False() */
(false)

/* exported Merge_Pdcl_Components(pdcl_components pc_1, pdcl_components pc_2) */
()
with (pc_1)
{ PdniNil; pc_2,
  PdniPair(h_1, t_1);
with(h_1)
{ NoComponent::Merge_Pdni_Components(t_1, pc_2),
  Data(head_id_1, '*', '*');
  with (pc_2)
  { PdniNil; pc_1,
    PdniPair(h_2, t_2);
    with (h_2)
    { NoComponent::Merge_Pdni_Components(pc_1, t_2),
      Data(head_id_2, '*', '*'),
      {((head_id_1 <= head_id_2) ? h_1::Merge_Pdni_Components(t_1, pc_2) : h_2::Merge_Pdni_Components(pc_1, t_2)),
       Op('*, '*', '*){h_1::Merge_Pdni_Components(t_1, pc_2)}
      }
    }
    Op(head_id_1, '*', '*'),
    with (pc_2)
    { PdniNil; pc_1,
      PdniPair(h_2, t_2);
      with (h_2)
      { NoComponent::Merge_Pdni_Components(pc_1, t_2),
        Data('*, '*', '*){h_2::Merge_Pdni_Components(pc_1, t_2)},
        Op(head_id_2, '*', '*'),
        {((head_id_1 <= head_id_2) ? h_1::Merge_Pdni_Components(t_1, pc_2) : h_2::Merge_Pdni_Components(pc_1, t_2))
      }
      }
    }
  }
}

/**-----------------------------*/
@pndi_components exported Sort_Pdni_Components(pndi_components pc)
{
  with (pc)
  { PdniNil; PdniNil,
    PdniPair(head, tail):Insert_Component(
      head,
      Sort_Pdni_Components(tail)
    )
  }
}

/**-----------------------------*/
@pndi_components exported Insert_Component(component h, pndi_components pc)
{
  with (pc)
  { PdniNil; PdniPair(h, PdniNil),
    PdniPair(head, tail):
    with (head)
    { NoComponent::Insert_Component(h, tail),
      Data(head_id, '*', '*');
      with (h)
      { NoComponent:pc,
        Data[head_id, '*', '*){((head_id <= head_id) ? h:pc : head::Insert_Component(h, tail))
      }
      }
    }
  }

/**-----------------------------*/
@t_operspec Find_T_op_spec_in_Data(id i, component d)
{
  with (d)
  { NoComponent::TopSpecNil,
    Data(i, ts, ti):
    with (ts)
    { TypeSpec('*, '*', oo, oo, oo, oo):Find_T_op_spec(i, oo)
    },
    Op('*, *, *)::TopSpecNil
  }
  */
/**-----------------------------*/

Op('*, *, *):head::Insert_Component(h, tail)
}
Op(head_id, *, '*'),
with(h)
{ NoComponent: pc,
  Data(head_id, '*', '*'):h:pc,
  Op(head_id, '*', '*'):((head_id <= head_id) ? h:pc : head::Insert_Component(h, tail))
}
APPENDIX D - Auxiliary Functions

t_oper_spec exported Find_T_Op_Spec(id i, o_operators oo)
{ }
    with (oo)
    { OperatorNil:topSpecNil,
        OperatorPair(tos, tail): with (tos)
            { topSpecNil:Find_T_Op_Spec(i, tail),
                topSpec(t_id, oo):(i == t_id)
                ? tos
                : Find_T_Op_Spec(i, tail)
            }
    }
}

/*---------------------------------------------------------------*/
t_oper_spec exported Make_TopSpecNil()
{ topSpecNil }

/*---------------------------------------------------------------*/
operator_impl Find_Operator_Impl(id i, operator_impl_list oll)
{ with (oll)
    { OpImplListNull:opImplNull,
        OpImplListPair(toi, tail): with (toi)
            { OpImplNull:Find_Operator_Impl(i, tail),
                OpImpl(t_id, oll):(i == t_id)
                ? oll
                : Find_Operator_Impl(i, tail)
            }
    }
}

/*---------------------------------------------------------------*/
IdSet Extract_Edge_Id_Set(operator_impl oll)
{ with (oll)
    { OpImplNull:IdSetNil,
        AddOpImpl('*'):IdSetNil,
        OperatorImpl(q, '+', '*'),
            with (q)
                { GraphNull:IdSetNil,
                    Graph('+, el'):Make_Edge_Id_Set{el}
                }
    }
}

/*---------------------------------------------------------------*/
IdSet Make_Edge_Id_Set(edge_list el)
{ with (el)
    { EdgeListNull:IdSetNil,
        EdgeListPair(ae, tail): with (ae)
            { AnEdgeNull:Make_Edge_Id_Set{tail},
                AnEdge(e_i, '+', '*'),
                    with (e_i)
                        { IdNull: Make_Edge_Id_Set{tail},
                            Id('*'): IdSetUnion(
                                SingletonIdSet(e_i),
                                Make_Edge_Id_Set{tail})
                        }
            }
    }
}

/*---------------------------------------------------------------*/

/** This third set of functions were already existing and
 **** for generating an operator head_node and its operator record.
 **** Documented originally as file: ed2.scl

---------------------------------------------------------------*/

OPNodePTR Make_Operator_Node;
    name, optional_type_name, oper_name,
    parameter_list, id, met, X, Y, radius, color,
    name_font, name_x, name_y,
    met_font, met_x, met_y,
    is_deleted, is_new, is_composite,
    is_terminator, is_modified)

char
    *name,
    *optional_type_name,
    *oper_name,
    *parameter_list;

OP_ID
    Id;

int
    met,
    X,
    Y,
    radius,
    color,
    name_font,
    name_x,
    name_y,
    met_font,
    met_x,
    met_y;

BOOL
    is_deleted,
    is_new,
    is_composite,
    is_terminator,
    is_modified;

{ OPNodePTR p;
    OPERATOR q;

#ifdef GRAPHICS_DEBUG
    /* debugging */
    print("Entering Make_Operator_Node\n");
#endif

#ifdef SDE_DEBUG
    print(" Make_Operator_Node: name = %s
", name);
    print(" Make_Operator_Node: optional_type_name = %s
", optional_type_name);
    print(" Make_Operator_Node: oper_name = %s
", oper_name);
    print(" Make_Operator_Node: parameter_list = %s
", parameter_list);
#endif

    p = (OPNodePTR)malloc(sizeof(OP_HEAD));
    q = (OPERATOR)malloc(sizeof(OP_NODE));
    p->op = q;
    p->next = NULL;

143
APPENDIX D - Auxiliary Functions

```c
q->name = strdup(name);
q->optional_type_name = strdup(optional_type_name);
q->operator_name = strdup(operand_name);
q->parameter_list = strdup(parameter_list);
q->name_font = name_font;
q->name_x = name_x;
q->name_y = name_y;
q->id = id;
q->met = met;
q->met_font = met_font;
q->met_x = met_x;
q->met_y = met_y;
q->X = X;
q->Y = Y;
q->radius = radius;
q->color = color;
q->is_deleted = is_deleted;
q->is_new = is_new;
q->is_composite = is_composite;
q->is_terminator = is_terminator;
q->is_modified = is_modified;

return (p);

TYPE_LIST Make_Type_Node(name)
char *name;
{
    TYPE_LIST h;
    PRODINSTANCE
    NullSet();
    h = (TYPE_LIST)malloc(sizeof(TYPE_NODE));
    h->type_name = strdup(name);
    h->undefined_op_impl = NullSet();
    h->base_type = Global_Type_List;
    Global_Type_List = h;
    return(h);
}

HeadPtr Make_Operator_Header(name, stream_list, operator_list, key, prod_no, del_flag)
char *name;
ST_PTR stream_list;
OPModePtr operator_list;
int key;
int prod_no;
{
    int op_id_count;
    HeadPtr h;
    PRODINSTANCE

    #ifdef SDE_DEBUG
    /* debugging */
    printf("Entering Make_Operator_Header\n");
    printf("Enter Name: %s\n", name);
    printf("Enter Key: %d\n", key);
    printf("Enter Prod No: %d\n", prod_no);
    
    h = (HeadPtr)malloc(sizeof(HEADER_NODE));
    h->name = strdup(name);
    h->operator_list = operator_list;
    h->op_id_no = key;
    h->prod_no = prod_no;
    h->met = met;
    h->is_composite = is_composite;
    h->is_terminator = is_terminator;
    h->is_modified = is_modified;
    
    return (h);
    #endif
```

144
FOREIGN C_INT_to_STR0(I)
int I;
{
    char buf[20];
    sprintf(buf, "%d", I);
    return (Str(str_to_str0(buf)));
}

/*---------------------------------------------*/
// The following function uses 'the_operator_list' and 'the_stream_list'
// and builds the SSL terms based on the 'live' nodes. The following can
// be built from the c-data structures:
//      Operator Specification (in case an operator is new)
//      Graph
//      Data stream declarations

Only those fields represented in the data structure are relevant to
this construction. Optional phyla and fields whose values cannot be
deduced from the data structures are represented by their NULL
constructors.

declrs.h contains the necessary macros for building the terms. This file
is generated by the SSL and erased after code generation.

/*---------------------------------------------*/

/* This function creates a graph with the new data stored in the c-data
   data structure. */

/*---------------------------------------------*/

PROD_INSTANCE Make_Graph(o_list, s_list)
    ONodePTR c_list;
    ST_PTR
    s_list;
{
    HeadPTR
        Find_Header_Node_from_Interp or TopImpl or TopSpec(),
            h;

    PROD_INSTANCE
        Make_Vertex_List(),
        Make_Edge_List(),
        graph,
        vertex_list,
        edge_list;

    boolean
t_bool;
#endif SDR_DEBUG1
/* debugging */
printf("Entering Make_Graph\n");
#endif

/* Generate Vertex List */
vertex_list = Make_Vertex_List(o_list);

/* Generate Edge List */
edge_list = Make_Edge_List(s_list);

/* Generate the Graph */
graph = Return_Graph(Graph(vertex_list, edge_list));

#endif SDR_DEBUG1
/* debugging */

printf("\n");
Print_Graph(o_list);
Print_Graph(s_list);
printf("\n");
return (graph);

/*---------------------------------------------*/

OMNodePTR Sort_Operator_List(q)
OMNodePTR q;
{
    ONodePTR
        head,
        tail;
    boolean
        done;

#ifdef SDR_DEBUG1
/* debugging */
printf("Entering Sort_Operator_List\n");
#endif

if (q == NULL)
{
    return(q);
}
else
{
    if (q->next == NULL)
        return(q);
    else
    {
        tail = Sort_Operator_List(q->next);
        if (strcmp(q->op->name, tail->op->name) <= 0)
            q->next = tail;
        return(q);
    }
else
{
    head = tail;
    done = false;
    while (tail->next != NULL && !done)
    {
        if (strcmp(q->op->name, tail->next->op->name) <= 0)
            done = true;
        else
            { tail = tail->next;
            }
APPENDIX D - Auxiliary Functions

```c
)
q->next = tail->next;
tail->next = q;
return(head);
)}
)
)/*----------------------------------------*/
ST_PTR Sort_Stream_List(p)
ST_PTR p;
{ST_PTR head, tail; boolean done;

#ifdef SDE_DEBUG_L /* debugging */
  printf("Entering Sort_Stream_List\n");
#endif

  if (p == NULL)
    return(p);
  else
    if (p->next == NULL)
      (return(p);
    else
      tail = Sort_Stream_List(p->next);
      if (strcmp(p->st->name, tail->st->name) <= 0)
        {p->next = tail;
         return(p);
        }
    else
      {head = tail;
       done = false;
        while (tail->next != NULL && !done)
          {if (strcmp(p->st->name, tail->next->st->name) <= 0)
            {done = true;
             }
          else
            {tail = tail->next;
             }
          }p->next = tail->next;
tail->next = p;
return(head);
)*
/* This function *sweeps* through the list of operators and builds a list
of vertices of the PSDL graph. Only operators that are live are placed
in the new PSDL vertex list.

Note: The following code ASSUMES
[1] AVertex(operator_id optional_time), OperatorId(operator_id_pairs)
    where operator_id_pairs always equal to OperatorIdPairsNull()
[2] time units are always in MS.

*/
PROD_INSTANCE Make_Vertex_List(p)
{OPNodePTR p;
{ OPERATOR q;
  PROD_INSTANCE /*
     name,
     */
    front_part, middle_part, suffix_part,
    make_operator_id_pairs(),
    time,
    vertex;
int Get_Unique_Id();

  char *get_vertex_type_name(),
    *get_vertex_oper_name(),
    *get_vertex_parameters(),
    *optional_type_name,
    *opser_name,
    *parameter_list,
    *clear_name,
    *remove_blanks_from_string(),
    dummy_name[100];
#ifdef SDE_DEBUG_L /* debugging */
  printf("Entering Make_Vertex_List\n");
#endif

  if (p == NULL)
    {return(VertexListNull);
    }
  else
    {q = p->op; /* q points to the operator's record */
    if (strcmp(q->name, "") == 0)
      
```
APPENDIX D - Auxiliary Functions

```c
/* need to create dummy name */
printf(dummy_name, "NO_NAME_4D", Get_Unique_Id());
q->name = strdup(dummy_name);
}
else {
    clean_name = remove_blanks_from_string(q->name);
    if (strcmp(clean_name, q->name) != 0)
        free(q->name);
        q->name = clean_name;
    else
        free(clean_name);

    if (q->is_deleted != TRUE) /* Make up the Vertex for this Operator */
        {
            if (q->is_new)
                {
                    q->id = Get_Unique_Id();
                    q->is_new = false;
                    q->optional_type_name = get_vertex_type_name(q->name);
                    q->operator_name = get_vertex_operator_name(q->name);
                    q->parameter_list = get_vertex_parameters(q->name);
                    if (strcmp(q->parameter_list, "") == 0)
                        {
                            if (strcmp(q->optional_type_name, "") != 0)
                                {
                                    /* there is a syntax error in the parameter list */
                                    free(q->optional_type_name);
                                    q->optional_type_name = strdup("*");
                                    free(q->name);
                                    q->name = strdup(q->operator_name);
                                }
                        }
                        else
                            {
                                if (strcmp(q->optional_type_name, "") == 0)
                                    {
                                        /* missing optional_type_name */
                                        free(q->parameter_list);
                                        q->parameter_list = strdup("*");
                                        free(q->name);
                                        q->name = strdup(q->operator_name);
                                    }
                    }
                    else
                        {
                            snprintf(dummy_name, "%s\tekke%\%a", q->optional_type_name, ",", q->operator_name, ",", q->parameter_list, ");
                            if (strcmp(dummy_name, q->name) != 0)
                                {
                                    /* vertex name differs from the original one */
                                    q->optional_type_name = get_vertex_type_name(q->name);
                                    q->operator_name = get_vertex_operator_name(q->name);
                                    q->parameter_list = get_vertex_parameters(q->name);
                                }
                        }
                        else
                            {
                                /* create the necessary phylum for the operator */
                                if (strcmp(q->optional_type_name, ";") != 0)
                                    front_part = OptionalTypedefId_SSLString(q->optional_type_name));
                                else
                                    front_part = OptionalTypedefNull;
                                if (strcmp(q->operator_name, ";") != 0)
                                    middle_part = Id_SSLString(q->operator_name);
                                else
                                    middle_part = IdNull;
                                printf("Make_Vertex_List: parameter_list = %s
", parameter_list);
                                /*
                                if (strcmp(q->parameter_list, ";") != 0)
                                    suffix_part = make_operator_id_pairs(q->parameter_list);
                                else
                                    suffix_part = OperatorIdPairNull;
                                    time = OptionalTimeNull;
                                else
                                    time = OptionalTime(Time(IntegerVal(C_INT_TO_STR(p->met)), UnitMS));
                                vertex = AVertex(OperatorId(front_part, middle_part, suffix_part),
                                #ifdef SDE_DEBUG_2
                                printf("Make_Vertex_List: operator = %s, id = %d\n", q->name,
                                (int)vertex);
                                #endif
                                return(VerteixListPair(vertex, Make_Vertex_List(p->next)));
                                else
                                    return( Make_Vertex_List(p->next) );
                                }
                            }
                        }
                    }
                }
        }
```

147
APPENDIX D - Auxiliary Functions

```c
return(AIDPair(AID, AIDLIST));
}

/* This function "sweeps" through the STREAM list and generates the PSDL graph's edge list. */
/*----------------------------------------------------------------------------*/
PROD_INSTANCE Make_Edge_List(p)
ST_PTR p;
{
    PROD_INSTANCE
    Get_Operator_Id_Name(),
    vl_id,
    vl_front_part,
    vl_middle_part,
    vl_suffix_part,
    v2_id,
    v2_front_part,
    v2_middle_part,
    v2_suffix_part,
    stream_name,
    stream_time;
    STREAM
    q;
    ORNodePTR
    r1;
    OPERATOR
    a1;
    char
    dummy_name[100],
    *dummy_name_ptr,
    *optional_type_name,
    *operand,
    *parameter_list,
    *clean_name,
    *remove_blanks_from_string(),
    *get_vertex_type_name(),
    *get_vertex_oper_name(),
    *get_vertex_parameters();

#define SDE_DEBUG1
    /* debugging */
    printf("Entering Make_Edge_List\n");
#endif
    if (p == NULL)
        return (EdgeListNil);
    else
        {
            j = 1;
            while ( (j) >= 48 && (j) <= 57 ) /* a decimal digit */
                [j] > 65 && (j) <= 90 ) /* an upper case */
                    (j) == 95 ) /* a '.' char */
                (j) == '\n' ) /* a null char */
            [j++] = 0;
        }
        if (j) == 0)
            AIDLIST = AIDNil;
        else
            AIDLIST = Extract AloneIdList From String(j+1);
        (j) = '\n';
        AID = Id(SSLstring(i));
```

APPENDIX D - Auxiliary Functions

```c
ifdef SDE_DEBUG
#endif

/* repeat the process for the second vertex */
if (q->to != NULLL)
{
    r1 = q->to;
    s1 = r1->op;
    
    /* update q's id if q is a new edge */
    if (q->is_new)
    {
        q->id = GetUniqueId();
        q->is_new = FALSE;
    }

    /* Get the first operator's name */
    if (q->from != NULL)
    {
        r1 = q->from; /* r1 points to the edge's first vertex head node*/
        s1 = r1->op; /* s1 points to the 'from' vertex's actual record */
    
    ifdef SDE_DEBUG
    endif

    clean_name = remove_blanks_from_string(s1->name);
    optional_type_name = get_vertex_type_name(clean_name);
    if (strcmp(optional_type_name, "") != 0)
    {
        v1_front_part = OptionalTypeId(id(SSString{
            optional_type_name}));
    }

    v1_front_part = OptionalTypeIdNull;

    oper_name = get_vertex_oper_name(clean_name);
    if (strcmp(oper_name, "") != 0)
    {
        v1_middle_part = Id(SSString(oper_name));
        v1_middle_part = IdNull;
    }

    parameter_list = get_vertex_parameters(clean_name);
    if (strcmp(parameter_list, "") != 0)
    {
        v1_suffix_part = make_operator_id_pairs(parameter_list);
    }

    else
        v1_suffix_part = OperatorIdPairsNull;

    free(clean_name);
}
else
{
    v1_front_part = OptionalTypeIdNull;
    v1_middle_part = Id(SSString("EXTERNAL"));
    v1_suffix_part = OperatorIdPairsNull;
}

free(clean_name);
}
```

```c
ifdef SDE_DEBUG
#endif

/* repeat the process for the second vertex */
if (q->to != NULLL)
{
    r1 = q->to;
    s1 = r1->op;
    
    /* update q's id if q is a new edge */
    if (q->is_new)
    {
        q->id = GetUniqueId();
        q->is_new = FALSE;
    }

    /* Get the first operator's name */
    if (q->from != NULL)
    {
        r1 = q->from; /* r1 points to the edge's first vertex head node*/
        s1 = r1->op; /* s1 points to the 'from' vertex's actual record */
    
    ifdef SDE_DEBUG
    endif

    clean_name = remove_blanks_from_string(s1->name);
    optional_type_name = get_vertex_type_name(clean_name);
    if (strcmp(optional_type_name, "") != 0)
    {
        v1_front_part = OptionalTypeId(id(SSString{
            optional_type_name}));
    }

    v1_front_part = OptionalTypeIdNull;

    oper_name = get_vertex_oper_name(clean_name);
    if (strcmp(oper_name, "") != 0)
    {
        v1_middle_part = Id(SSString(oper_name));
        v1_middle_part = IdNull;
    }

    parameter_list = get_vertex_parameters(clean_name);
    if (strcmp(parameter_list, "") != 0)
    {
        v1_suffix_part = make_operator_id_pairs(parameter_list);
    }

    else
        v1_suffix_part = OperatorIdPairsNull;

    free(clean_name);
}
else
{
    v1_front_part = OptionalTypeIdNull;
    v1_middle_part = Id(SSString("EXTERNAL"));
    v1_suffix_part = OperatorIdPairsNull;
}

free(clean_name);
```
APPENDIX D - Auxiliary Functions

extern boolean Global_Enforce_Constistency;

extern PRODUCTION
prod_prot,
prod_pedi_pair,
prod_op,
prod_data,
prod_op_spec,
prod_op_impl,
prod_t_op_impl,
prod_input_list,
prod_inputs,
prod_output_list,
prod_outputs,
prod_type_decl,
prod_graph,
prod_a_decl,
prod_decl,
prod_stream,
prod_cc,
prod_constraints,
prod_a_constraint;

ATREE
atree = bu_atree(br_buf(cur_browser));

PRODUCTION
top_production,
marker_production;

PROD_INSTANCE
proto,
P,
temp_p,
top_p,
marker_id,
component_id,
op_id,
t_op_impl_id,
Make_IdNull(),
IsTypeDeclNull(),
IsAConstraintNull(),
Get_Id_From_Inputs_list(),
Get_Id_From_Outputs_list(),
Get_Id_From_Stream(),
Get.Operator_Id_From_Constraints(),
Get_Id_From_Type_Decl(),
Find_O_Inputs(),
Find_O_Outputs(),
Find_Stream_Type_Decl(),
Find_A_Constraint(),
Find_T0pImpl_in_operator_Impl_list();

HeadPtr
h,
Find_Header_Node_from_Op_or_T0pImpl_or_T0pSpec(),
Make_Operator_Header();

LINKED_LIST
tmp_head,
current_pos_trace = NULL;
APPENDIX D - Auxiliary Functions

```c
void
Enforce_Consistency(),
Link_To_Structure(),
Free_Linked_List();

char
* name;

#ifdef SDE_DEBUG
*/debugging*/
printf("Entering House_Cleaning\n");
#endif

if (Global_Enable_Consistency)
{
    p = selection_apex(new_selection);
    /* remember the cursor position before fixing the a_tree */
    /* the possible phyhum checkpoints are: Prot, Op, Data. */
    /* InputsListPair, OutputsListPair, Steam, ConstraintsPair */
    temp_p = p;
    current_pos_trace = NULL;
    while (list_top(temp_p) && (production(temp_p) != prod_op)
           && (production(temp_p) != prod_data)
           && (production(temp_p) != prod_t_op_impl))
    {
        temp_p = temp_p->next;
        current_pos_trace = temp_head;
    }
#endif

#ifdef SDE_DEBUG
*/debugging*/
printf("House_Cleaning: son_number = %d\n", son_number(temp_p));
#endif

    temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
    temp_head->item_number = son_number(temp_p);
    temp_head->next = current_pos_trace;
    current_pos_trace = temp_head;
    temp_p = father(temp_p);

    top_p = temp_p;

    if (production(top_p) == prod_data)
    {
        top_production = prod_data;
        component_id = id_from_Data(top_p);
        Free_Linked_List(current_pos_trace);
    }
    else if (production(top_p) == prod_op)
    {
        top_production = prod_op;
        component_id = id_from_Op(top_p);
        Free_Linked_List(current_pos_trace);
    }
    else
    {
        if (FirstChild(component_id) == 0)
        {
            h = Find_Header_node_from_Op_or_TopImpl_or_TopSpec(top_p);
            if (h == NULL)
            {
                printf("House_Cleaning: cannot find header_node\n");
            }
        }
        else
        {
            h = Make_Operator_Header(str_split_to_str(int_value(Get_Id(id_from_Op(top_p)))),
                                      NULL, NULL, Get_Undefined_Id(), (int)top_p, FALSE);
            Link_To.Structure(h);
        }
        if (!h->multi_op_error)
        {
            temp_p = p;
            while (list_top(temp_p) != prod_input_list)
            {
                temp_p = temp_p->next;
            }
            if (production(temp_p) == prod_input_list && h->input_error)
            {
                printf("House_Cleaning: production(temp_p) == prod_input_list\n");
            }
            Free_Linked_List(current_pos_trace);
            marker_production = prod_input_list;
            Free_Linked_List(current_pos_trace);
            Free_Linked_List(current_pos_trace);
        }
    }
#endif

    Free_Linked_List(current_pos_trace);
```

152
APPENDIX D - Auxiliary Functions

#define SDE_DEBUG_3

ifdef SDE_DEBUG_3
    printf("House Cleaning: pass Free_Linked_List\n");
endif

marker_production = prod_stream;

    temp_p = p;
    while (!At_Tcp(temp_p) && (production(temp_p) != prod_type_decl))
    {
        temp_p = father(temp_p);
    }
#endif SDE_DEBUG_3

ifdef SDE_DEBUG_3
    printf("House Cleaning: pass while loop\n");
endif

if [production(temp_p) == prod_type_decl] { marker_id = Get_Id_From_Type_Decl(temp_p); }
#endif SDE_DEBUG_3

ifdef SDE_DEBUG_3
    printf("House Cleaning: marker_id = %d", str0_to_str_ro(StValue(Get_Id(marker_id))));
#endif

e else
{
    marker_id = Make_IdNull();
}
#endif SDE_DEBUG_3

ifdef SDE_DEBUG_3
    printf("House Cleaning: marker_id = IdNull\n");
endif

ifdef SDE_DEBUG_3
    printf("House Cleaning: pass Get_Id_From_Type_Decl\n");
endif

} else if ([production(temp_p) == prod_constraints] && h->constraint_error)
{
#undef SDE_DEBUG_3
    /*debugging*/
    printf("House Cleaning: production[temp_p] == prod_constraints\n");
#endif

Free_Linked_List(current_pos_trace);
marker_production = prod_constraints;
marker_id = Get_Operator_Id_From_Constraints(p);
} else
{
    marker_production = top_production;
}
}
#endif SDE_DEBUG_3

ifdef SDE_DEBUG_3
    printf("House Cleaning: production[temp_p] == prod_t_op_impl\n");
endif

top_production = prod_t_op_impl;

t_op_impl_id = id_from_TOpImpl(top_p);
#endif SDE_DEBUG_3

ifdef SDE_DEBUG_3
    printf("House Cleaning: t_op_impl_id = %d", str0_to_str_ro(StValue(Get_Id(t_op_impl_id))));
#endif
/* get component_id */

    temp_p = top_p;
    while (production(temp_p) != prod_data)
    {
        temp_p = father(temp_p);
    }
    component_id = id_from_Data(temp_p);
#endif SDE_DEBUG_3

ifdef SDE_DEBUG_3
    printf("House Cleaning: component_id = %d\n", str0_to_str_ro(StValue(Get_Id(component_id))));
#endif

/* get header node */

if [IntValue(!isNull(t_op_impl_id)) == 0] { h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(top_p); }

if (h == NULL)
{

#endif SDE_DEBUG_3

ifdef SDE_DEBUG_3
    /*debugging*/
    printf("House Cleaning: cannot find header_node\n");
#endif

} else
{
    /* need to make new header node */

    h = Make_Operator_Header(str0_to_str_ro(StValue(Get_Id(id_from_TOpImpl(top_p))))),
        NULL, NULL, Get_Unique_Id(),
    (int)top_p, FALSE);

    Link_To_Structure(h);
}
#endif SDE_DEBUG_3

/* see if cursor is positioned at streams or constraints */

    temp_p = p;
    while ((temp_p != top_p) && (production(temp_p) != prod_stream)
        && (production(temp_p) != prod_constraints))
    {
        temp_p = father(temp_p);
    }
#endif SDE_DEBUG_3

if ([production(temp_p) == prod_stream] && h->stream_error)
{
#undef SDE_DEBUG_3
    /*debugging*/
    printf("House Cleaning: production[temp_p] == prod_stream\n");
#endif

Free_Linked_List(current_pos_trace);
#endif SDE_DEBUG_3

ifdef SDE_DEBUG_3
    printf("House Cleaning: pass Free_Linked_List\n");
#endif

ifdef SDE_DEBUG_3
    printf("House Cleaning: pass Free_Linked_List\n");
#endif

temp_p = p;
while ((temp_p != top_p) && (production(temp_p) != prod_type_decl))
{
    temp_p = father(temp_p);
}
#endif SDE_DEBUG_3

153
if (production(temp_p) == prod_type_decl)
    marker_id = Get_Id_From_Type_Dec(temp_p);

str0_to_str_ro(StrValue(Get_Id(marker_id)));
#endif
else
    marker_id = Make_IdNull();
#endif

if (production(temp_p) == prod_constraints)
    /*debugging*/
    printf("House_Cleaning: production(temp_p) == prod_constraints\n");
#endif
else
    marker_production = top_production;

if (top_production == top_data)
    if (prod_constraints & (h->multi_op_error))
        /*debugging*/
        printf("House_Cleaning: production(temp_p) == prod_constraints\n");
#endif
else
    marker_production = top_production;
else
    marker_production = top_production;

Free_Linked_List(current_pos_trace);
marker_production = prod_constraints;
marker_id = Get_Operator_Id_From_Constraints(p);
}
else
    marker_production = top_production;

Free_Linked_List(current_pos_trace);
}
else
    marker_production = top_production;

Enforce_Constistency();
#endif

if (IntValue(IdsNull(component_id)) == 0)
    /* reset cursor position */
    if (top_production == prod_data)
        if (top_production == prod_op)
            if (top_production == prod_t_op_impl)
                top_p = Find_Component(component_id, pdml_components_from_Prot(Global_Protos));

#endif

if (production(temp_p) == prod_type_decl)
    marker_id = Get_Id_From_Type_Dec(temp_p);

str0_to_str_ro(StrValue(Get_Id(component_id)));
#endif
else
    marker_id = Make_IdNull();
#endif

if (top_production == top_data)
    if (top_production == prod_op) & (h->multi_op_error))
        /*debugging*/
        printf("House_Cleaning: component_id = %s\n",
            str0_to_str_ro(StrValue(Get_Id(component_id))));
#endif

if (top_production == top_data)
    if (prod_constraints & (h->multi_op_error))
        /*debugging*/
        printf("House_Cleaning: component_id = %s\n",
            str0_to_str_ro(StrValue(Get_Id(component_id))));
        /*
        str0_to_str_ro(StrValue(Get_Id(son_top_p, 1))));
        */
#endif

if (marker_production == top_production)
    temp_p = top_p;

if (marker_production == top_production)
    /*debugging*/
    printf("House_Cleaning: marker_production == top_production\n");
#endif

temp_head = current_pos_trace;
while (temp_head != NULL)
    temp_p = son(temp_p, (temp_head->item_number));
    temp_head = temp_head->next;
    p = temp_p;
    Free_Linked_List(current_pos_trace);
    if (marker_production == prod_input_list)
        /*debugging*/
        printf("House_Cleaning: marker_production == prod_input_list\n");
#endif

/*
 temp_p = son(son(temp_p, 2), 2);
*/

if (IntValue(IdsNull(marker_id)) == 0)
    /*debugging*/
    printf("House_Cleaning: marker_id = %s\n",
        str0_to_str_ro(StrValue(Get_Id(component_id))));
#endif

p = Find_D_Inputs(marker_id, temp_p);
if (BoolValue(IsTypeDeclNil(p))
APPENDIX D - Auxiliary Functions

```c

Else if (marker_production == prod_output_list)
{
#endif

/*
 temp_p = s_n(son(temp_p, 2), 3);
*/
 temp_p = c_o_outputs_list_from_OperatorSpec(
 operator_spec_from_op(temp_p));
 if ([IntValue(IdIsNull(marker_id)) == 0])
{
#endif

/*
 House_Cleaning: marker_id = %s
 str0_to_str_ro(St rval(Get_Id(marker_id)));
*/

 p = Find_O_Outputs(marker_id, temp_p);

#endif

/*
 House_Cleaning: returning from Find_O_Outputs
*/

 if (BooValue(FirstnameDeclNull(p)))
{
 p = temp_p;
 }
 else
 {
 p = temp_p;
 }

#endif

Else if (marker_production == prod_stream)
{
#endif

/*
 House_Cleaning: marker_production == prod_stream
*/

 temp_p = s_n(son(son(temp_p, 3), 2), 1);
 temp_p = o_n(outputs_list_from_Declarations(
 declarations_from_OperatorImpl(
 operator_impl_from_op(temp_p)));
 if ([IntValue(IdIsNull(marker_id)) == 0])
{
#endif

/*
 House_Cleaning: marker_production == prod_output_list
*/

 p = Find_Streams_Type_Decl(marker_id, temp_p);
 if (BooValue(FirstnameDeclNull(p)))
{
 p = temp_p;
 }
 else
 {
 p = temp_p;
 }

#endif

Else if (marker_production == prod_constraints)
{
#endif

/*
 House_Cleaning: marker_production == prod_constraints
*/

 temp_p = constraints_from_CC(
 cc_from_OperatorImpl(
 operator_impl_from_op(temp_p)));
 if ([IntValue(OperatorIdIsNull(marker_id)) == 0])
{
#endif

/*
 House_Cleaning: marker_id = %s
 str0_to_str_ro(St rval(Get_Id(marker_id)));
*/

 p = Find_CCONSTRAINT(marker_id, temp_p);
 if (BooValue(FirstnameDeclNull(p)))
{
 p = temp_p;
 }
 else
 {
 p = temp_p;
 }

#endif

Else if (top_production == prod_t_op_impl)
{
#endif

/*
 House_Cleaning: after Enforce_Constancy
*/

 printf("House_Cleaning: top_production == prod_t_op_impl\n");
 printf("House_Cleaning: t_op_impl_id = %s\n", 
 str0_to_str_ro(St rval(Get_Id(t_op_impl_id))));

/*
 House_Cleaning: marker_production == prod_stream
*/

 p = Find_Streams_Type_Decl(marker_id, temp_p);
 if ([IntValue(IdIsNull(t_op_impl_id)) == 0])
{
 temp_p = Find_TopImpl_in_operator_impl_list(
 t_op_impl_id, 
 operator_impl_list_from_TypeImpl(
 type_impl_from_Data(top_p)));
 if (production(temp_p) == prod_t_op_impl)
{
#endif
```

155
print("House Cleaning: marker_id = \$s\n",
str0_to_str_ro(StrValue(Get_Id(id_from_OperatorId(marker_id)))));
#endif

if (marker_production == top_production)
{
    #ifdef SDE_DEBUG_3
        /*debugging*/
        printf("House Cleaning: marker_production == top_production\n");
    #endif

    temp_head = current_pos_trace;
    while (temp_head != NULL)
    {
        temp_small = temp_head->item_number;
        temp_head = temp_head->next;
    }
    p = temp_p;
    Free_Linked_List(current_pos_trace);
}
else if (marker_production == prod_stream)
{
    #ifdef SDE_DEBUG_3
        printf("House Cleaning: marker_production == prod_stream\n");
    #endif

    temp_p = optional_stream_from_Declarations(
                declarations_from_OperatorImpl(
                        operator_impl_from_TopImpl(temp_p)));
    if (IntValue(IdIsNull(marker_id)) == 0)
    {
        /*debugging*/
        printf("House Cleaning: multi-op-error in \$s\n",
                str0_to_str_ro(StrValue(Get_Id(id_from_OperOp[top_p]))));
    }
    #ifdef SDE_DEBUG_3
        printf("House Cleaning: marker_id = \$s\n",
                str0_to_str_ro(StrValue(Get_Id(marker_id))));
    #endif

    p = Find_Stream_Type_Dec(marker_id, temp_p);
    if (BoolValue(IsTypeDeclNull(p))
    {
        p = temp_p;
    }
    else
    {
        p = temp_p;
    }
}
else if (marker_production == prod_constraints)
{
    #ifdef SDE_DEBUG_3
        /*debugging*/
        printf("House Cleaning: marker_production == prod_constraints\n");
    #endif

    temp_p = constraints_from_Cc(
                cc_from_OperatorImpl(
                        operator_impl_from_TopImpl(temp_p)));
    if (IntValue(IdIsNull(marker_id)) == 0)
    {
        /*debugging*/
        printf("House Cleaning: marker_id = \$s\n",
                str0_to_str_ro(StrValue(Get_Id(id_from_OperatorId(marker_id)))));
    }
    #ifdef SDE_DEBUG_3
        printf("House Cleaning: marker_id = \$s\n",
                str0_to_str_ro(StrValue(Get_Id(id_from_OperatorId(marker_id)))));
    #endif
    p = Find_A_Constraint(marker_id, temp_p);
    if (BoolValue(IsAConstraintNull(p))
    {
        p = temp_p;
    }
   }
else
    {
        p = temp_p;
    }
}

else
    {
        p = type_impl_from_Data(top_p);
    }
}

else
    {
        p = type_impl_from_Data(top_p);
    }
}else
    {
        p = Global_Prototype;
        Free_Linked_List(current_pos_trace);
    }
#endif

/* update tree, buffer and selection */
    temp_p = one_point_selection(p);
    move_selection(axe, temp_p);
    br_set_insert_pt_to_selection(cur_browser);
    cmd_cond_modifies(cur_browser, cur_buffer);
    br_paint_all();
}

/*----------------------------------------------*/
void Refresh_Graph_Viewer()
{
    HeadPtr
APPENDIX D - Auxiliary Functions

```c
h,
Find_Header_Node_from_Op_or_T0pImpl_or_T0pSpec();
Make_Operator_Header();

PROD_INSTANCE
IsI(Null),
component_id,
t_op_impl_id,
op_p,
t_op_impl_p,
temp_p;

void
Update_Operator();
Move_To_Structure_Front();

#endif SDE_DEBUG_3
printf("Entering Refresh_Graph_Vewriter\n");
#endif

temp_p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));
while ((lat_top(temp_p) && (production(temp_p) != prod_op) &&
       (production(temp_p) != prod_t_op_impl))
    { temp_p = father(temp_p);
    }

if (lat_top(temp_p))
{ if (production(temp_p) == prod_op)
  {
    op_p = temp_p;
    component_id = id_from_op(op_p);
    if (IntValue(IDInNull(component_id)) == 0)
      {
        h = Find_Header_Node_from_Op_or_T0pImpl_or_T0pSpec(op_p);
        if (h == NULL)
          {
#ifdef SDE_DEBUG_3
        "debugging"/  printf("Refresh_Graph_Vewriter: cannot find header node\n");
        #endif
        /* need to make new header node */
        Make_Operator_Header(str0_to_str_ro(StrValue(Get_Id(id_from_op(op_p))))),
        Link_To_Structure(h);
        */
      } /* to order the header nodes so that most-recently-used-first */
      Move_To_Structure_Front(h);
    /* see if need to refresh graph view */
    if (strcmp(str0_to_str_ro(StrValue(Get_Id(component_id))),
               Global_Current_Op_Name) != 0)
      {
        /* cursor is at a different operator */
        if (strcmp(Global_Current_Op_Name, "") != 0)
          { free(Global_Current_Op_Name);
            Global_Current_Op_Name = strdup(str0_to_str_ro(
              StrValue(Get_Id(id_from_op(op_p)))));
            /*
               */
            Global_Refresh_Graph_Vewriter = true;
          }
        else
          { temp_p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));
            while ((lat_top(temp_p) && (production(temp_p) != prod_graph)
               && (production(temp_p) != prod_graph_null))
              { temp_p = father(temp_p);
              }
            if ((production(temp_p) == prod_graph) ||
                (production(temp_p) == prod_graph_null))
              { /* cursor is at the same operator but graph is selected*/
                Global_Refresh_Graph_Vewriter = true;
              }
          }
        }
      } /* if (production(temp_p) == prod_op) */
    else
      { op_p = temp_p;
        t_op_impl_p = temp_p;
        t_op_impl_id = id_from_T0pImpl(t_op_impl_p);
        if (IntValue(IDInNull(t_op_impl_id)) == 0)
          {
            h = Find_Header_Node_from_Op_or_T0pImpl_or_T0pSpec(t_op_impl_p);
            if (h == NULL)
              {
#ifdef SDE_DEBUG_3
        "debugging"/  printf("Refresh_Graph_Vewriter: cannot find header node\n");
        #endif
        /* need to make new header node */
        Make_Operator_Header(str0_to_str_ro(StrValue(Get_Id(id_from_T0pImpl(t_op_impl_p))))),
        Link_To_Structure(h);
        */
          } /* to order the header nodes so that most-recently-used-first */
          Move_To_Structure_Front(h);
        /* see if need to refresh graph view */
        if (strcmp(str0_to_str_ro(StrValue(Get_Id(t_op_impl_id))),
                   Global_Current_Op_Name) != 0)
          { free(Global_Current_Op_Name);
            Global_Current_Op_Name = strdup(str0_to_str_ro(
              StrValue(Get_Id(id_from_T0pImpl(t_op_impl_p)))));
            /*
               */
            Global_Refresh_Graph_Vewriter = true;
          }
      }
    /* if (lat_top(temp_p)) */
  }
#endif
```
APPENDIX D - Auxiliary Functions

```c
/* cursor is at a different operator */
if (strcmp(Global_Current.Op_Name, "") != 0)
{
    free(Global_Current.Op_Name);
}
Global_Current.Op_Name = strdup(str_to_str_ro(ToStrValue(Get_Id(t_op_impl_id))));
Global_Refresh_Graph Viewer = true;
} else
{
    temp_p = selection_apex(SE_selection(bu_selection(bu_buf(cur_browser))));
    while ((temp_p->temp) && (production(temp_p) != prod_graph_null))
    {
        temp_p = father(temp_p);
    }

    if ((production(temp_p) == prod_graph) ||
        (production(temp_p) == prod_graph_null))
    {
        /* cursor is at the same operator but graph is selected*/
        Global_Refresh_Graph Viewer = true;
    }
}
}

if (Global_Refresh_Graph Viewer)
{
    Global_Refresh_Graph Viewer = false;
    Update_Operator(op_p, h);
    current_graph = h;
    the_operator_list = h->operator_list;
    the_stream_list = h->stream_list;
#endif //DEF DEBUG3
    printf("--OPERATOR FOUND IN THE LIST--
");
    printf("%s", h->name);
    Print_Operators_S_Operators(h->operator_list);
    Print_Operators_S_Streams(h->stream_list);
#endif //DEF DEBUG3
    refresh();
}
#endif //DEF DEBUG3
    printf("Leaving Refresh_Graph Viewer\n");
#endif

/*----------------------------------------*/
void Update_Operator(p, q)
PROD_INSTANCE
p;
HeadPtr
q;
int
*x_ptr, /* for the x coordinate of the new vertices when needed */
*y_ptr, /* for the y coordinate of the new vertices when needed */
{
    PROD_INSTANCE
    vertex_list,
    vertex,
    Get_Operator_Impl(),
    Get_Vertex_list(),
    Get_Vertex(),
    Rest_Of_Vertex_list();
    OPMODEPTR
    v;
#endif //DEF DEBUG1
    printf("Entering Update_Operator_List\n");
```
APPENDIX D - Auxiliary Functions

```c
#endif;

/* initialize the 'is_deleted' and 'is_modified' fields in all operators
 in q->operator_list */
if (q == NULL)
{
    v = NULL;
    v->is_deleted = TRUE;
    v->is_modified = FALSE;
}
else
{
    v = q->next;
}

if ((production(p) == prod_op)
    vertex_list = Get_Vertex_List(operator_impl_from_op(p));
else
    vertex_list = Get_Vertex_List(operator_impl_from_TocImpl(p));

while (production(vertex_list) == prod_vertex_list_pair)
{
    vertex = Get_Vertex(vertex_list);
    vertex_list = Rest_Of_Vertex_List(vertex_list);
    Update_Vertex(vertex, q, x_ptr, y_ptr);
}

/* update the x coordinate for the new operator if one is needed */
int *x_ptr, /* the x coordinate for the new operator if one is needed */
int *y_ptr; /* the y coordinate for the new operator if one is needed */

/* get 'is_composite' value from the corresponding op_node if one exists. */
if (Op_Name.Is_In_List(vertex_op_name))
{
    if (f == NULL)
    {
        /* need to create a new header for the vertex */
        f = Make_Operator_Header(vertex_op_name, NULL, NULL, Get_Unique_Id(),
                                  fake_op_no, fake_is_deleted);
        if (strcmp(vertex_type_name, "") != 0)
            f->type_id = strdup(vertex_type_name);
        Link_To_Structure(f);
    }
    if (strcmp(vertex_type_name, "") == 0)
    {
        is_composite = f->is_composite;
    }
    else
    { /* an instance of a type-operator */
        is_composite = false;
    }
    met = Get_Met(v);
    /* see if vertex is already in the operator_list */
    k = Vtx_Name.Is_In_List(vertex_type_name, vertex_op_name, vertex_parameters, h-
                            operator_list);
    if (k != NULL)
    {
        k->op->met = met;
        k->op->is_composite = is_composite;
        k->op->is_deleted = FALSE;
    }
    else
```

159
APPENDIX D - Auxiliary Functions

```c
else {
    /* new vertex to be inserted */
#endif GRAPHICS_DEBUG
    /* debugging */
    printf("Update_Vertex: create new vertex \"%
    new_vertex = Make_Operator_Node(dummy,
    vertex_type_name,
    vertex_name,
    vertex_parameters,
    GetUniqueId(),
    met,
    x_ptr,
    y_ptr,
    fake_radius,
    fake_color,
    fake_font, /* name_font */
    (*x_ptr + 10), /* name_x */
    (*y_ptr + 34), /* name_y */
    fake_font, /* met_font */
    (*x_ptr, /* met_x */
    (*y_ptr, /* met_y */
    fake_is_deleted,
    fake_is_new,
    is_composite,
    fake_is_terminator,
    fake_is_modified);

    Link_Operator(new_vertex, h);
    if (*x_ptr < 600)
        (*x_ptr = (*x_ptr + 80;
    if (((*x_ptr / 80) & 2) == 0)
        (*y_ptr = (*y_ptr + 50;
    else
        (*y_ptr = (*y_ptr - 40;
    )
    else
        (*y_ptr = (*y_ptr + 80;
    )
}

/--------------------------------------------------------------------------*/
Update_Stream_List(p, q, x_ptr, y_ptr)
PROD_INSTANCE
p;
HeadPtr
q;
int
*x_ptr, /* the x coordinate for the new operator if one is needed */
*y_ptr, /* the y coordinate for the new operator if one is needed */
{ PROD_INSTANCE
  temp_p = q->state_id_set;
  while ( (!BoolValue(IsNone(temp_p)))
          { printf("Update_Edge_List: state_id = \"%
            str_to_str_ro(StringValue(Get_Id(FistElement(temp_p))));
            temp_p = IdSetTail(temp_p);
          }
    }

/--------------------------------------------------------------------------*/
Update_Edge(a, h, x_ptr, y_ptr)
PROD_INSTANCE
a;
HeadPtr
h;
int
*x_ptr, /* the x coordinate for external label if one is needed */
*y_ptr, /* the y coordinate for external label if one is needed */
{ int
  latency,
  Get_Latency();
```

160
APPENDIX D - Auxiliary Functions

```c
char *Get_Parameters_From_Operator_Id_Pairs(),
*optional_type_name,
*operand_name,
*parameter_list,
*edge_name;

BOOL is_state;

ST_PTR Make_Stream_Node(),
new_edge;

STREAM
Edge_Name_Is_In_List(),
k;

SPLINE_PTR
spl_ptr;

OPNodePTR
Vtx_Name_Is_In_List(),
from,
to,
op_list;

void
Link_Stream();

int
from_x,
from_y,
to_x,
to_y;

PROD_INSTANCE
temp_p,
FirstElement(),
IdSetTail();

Get_Id();
Get_Edge_From_Vertex_Type_Id_Name(),
Get_Edge_From_Vertex_OpIdPairs(),
Get_Edge_To_Vertex_Type_Id_Name(),
Get_Edge_To_Vertex_OpIdPairs(),
from_vertex_id,
to_vertex_id,
IsElement();

#endif SDE_DEBUG_1
printf("Entering Update_Edge\n");
#endif

edge_name = strdup(STR_to_str_ro(ToStrValue(Get_Edge_Name(o))));
from_vertex_id = Get_Edge_From_Vertex_OpId(e);
to_vertex_id = Get_Edge_To_Vertex_OpId(e);

#endif SDE_DEBUG_3
printf("Update_Edge: edge name = %s\n", edge_name);
#endif

optional_type_name = STR_to_str_ro(ToStrValue(Get_Edge_From_Vertex_Type_Id_Name(from_vertex_id)));
#endif SDE_DEBUG_3
printf("Update_Edge: from vertex optional_type_name = %s\n", optional_type_name);
#endif

operand_name = STR_to_str_ro(ToStrValue(
Get_Edge_From_Vertex_OpIdPairs(from_vertex_id)));
#endif SDE_DEBUG_3
printf("Update_Edge: from vertex operand_name = %s\n", operand_name);
#endif

parameter_list = Get_Parameters_From_Operator_Id_Pairs(
Get_Edge_From_Vertex_OpIdPairs(from_vertex_id)));
#endif SDE_DEBUG_3
printf("Update_Edge: from vertex parameter_list = %s\n", parameter_list);
#endif

if (strcmp(operand_name, "EXTERNAL") != 0)
{
from = Vtx_Name_Is_In_List(optional_type_name, operand_name,
parameter_list, h->operator_list);
else
from = NULL;

if (from == NULL)
{
/* from EXTERNAL */
from_x = *x_ptr;
from_y = *y_ptr;
*y_ptr = *y_ptr + 40;
}
else
{
from_x = from->op->x;
from_y = from->op->y;
}

optional_type_name = STR_to_str_ro(ToStrValue(
Get_Edge_To_Vertex_Type_Id_Name(to_vertex_id)));
#endif SDE_DEBUG_3
printf("Update_Edge: to vertex optional_type_name = %s\n", optional_type_name);
#endif

operand_name = STR_to_str_ro(ToStrValue(
Get_Edge_To_Vertex_OpIdPairs(to_vertex_id)));
#endif SDE_DEBUG_3
printf("Update_Edge: to vertex operand_name = %s\n", operand_name);
#endif

parameter_list = Get_Parameters_From_Operator_Id_Pairs(
Get_Edge_To_Vertex_OpIdPairs(to_vertex_id)));
#endif SDE_DEBUG_3
printf("Update_Edge: to vertex parameter_list = %s\n", parameter_list);
#endif
```
if (strcmp(oper_name, "EXTERNAL") != 0)
    to = Vtx_Name_Is_In_List(optional_type_name, oper_name,
        parameter_list, h->operator_list);
else
    to = NULL;

if (to == NULL)
{
    /* to EXTERNAL */
    to_x = *x_ptr;
    to_y = *y_ptr;
    *y_ptr = *y_ptr + 40;
}
else
{
    to_x = to->op->x;
    to_y = to->op->y;
}

latency = Get_Latency(e);

#define SDR_DEBUG_3
temp_p = h->state_id_set;
while (!!BoolValue(IsNull(temp_p)))
{
    printf("Update_Edge: state_id = %s\n", tmp_to_str_ro(StrValue(Get_Id(FirstElement(temp_p)))));
    temp_p = IdDefTail(temp_p);
}
#endif

is_state = BoolValue(IsElement(id_from_AnEdge(e), h->state_id_set));

#define SDR_DEBUG_3
printf("Update_Edge: %s is_state = %s\n", id_from_AnEdge(e)),
    (is_state)?"TRUE":"FALSE");

temp_p = h->state_id_set;
while (!!BoolValue(IsNull(temp_p)))
{
    printf("Update_Edge: state_id = %s\n", tmp_to_str_ro(StrValue(Get_Id(FirstElement(temp_p)))));
    temp_p = IdDefTail(temp_p);
}
#endif

k = Edge_Name_Is_In_List(edge_name, from, to, h->stream_list);
if (k != NULL)
{
    k->latency = latency;
    k->is_state_variable = is_state;
    /*
     * k->from = from;
     * k->to = to;
     */
    k->is_deleted = FALSE;
}
else
{
    #ifdef GRAPHICS_DEBUG
    /* debugging */
    printf("edge_name = %s\n", edge_name);
    printf("from_x = %d, to_x = %d\n", from_x, to_x);
    #endif

    if ((from == NULL) || (to == NULL))
    {
        /* External streams must have at least one control point */
        spl_ptr = (SPLINE_PTR) malloc(sizeof(SPLINE_NODES));
        spl_ptr->next = NULL;
        if (from == NULL)
        {
            spl_ptr->x = from_x;
            spl_ptr->y = from_y;
        }
        else
        {
            spl_ptr->x = to_x;
            spl_ptr->y = to_y;
        }
    }
    else
    {
        spl_ptr = NULL;
    }

    /* need to create new stream node */
    new_edge = Make_Stream_Node(edge_name, Get_Unique_Id(),
        from,
        to,
        fake_font, /* name_font */
        (int)(from_x + to_x)/2, /* name_x */
        (int)(from_y + to_y)/2 + 30, /* name_y */
        fake_font, /* latency_font */
        (int)(from_x + to_x)/2, /* latency_x */
        (int)(from_y + to_y)/2 + 45, /* latency_y */
        spl_ptr,
        latency,
        fake_is_deleted,
        fake_is_new,
        fake_is_modified,
        is_state);

    Link_Stream(new_edge, h);
}
}

void Add_Input_Output_State_Nodes(h)
HeadPtr h;
{
    int input_count,
        output_count,
        state_count,
        first_x,
        node_x,
        edge_label_y,
        edge_y,
APPENDIX D - Auxiliary Functions

OPNode*PTR
Vtx_Name_Is_In_List(),
Make_Operator_Node(),
new_vertex,
k;
STREAM
Edge_Name_Is_In_List(),
t;
ST_PTR
new_edge,
Make_Stream_Node();
SPLINE_PTR
spl_ptr;
PROD_INSTANCE
temp_p,
Get_Id(),
IsNull(),
IDSetSize(),
FirstElement(),
IDSetTail();
char
*edge_name;
first_x = 0;
if (h != NULL)
{
    input_count = IntValue(IDSetSize(h->inh_input_id_set));
    output_count = IntValue(IDSetSize(h->inh_output_id_set));
    state_count = IntValue(IDSetSize(h->state_id_set));
    if (input_count > 0)
    {
        /* add the new vertex */
        node_x = first_x + (80 * (input_count - 1) / 2);
        k = Vtx_Name_Is_In_List('"', 'INPUT', '"', h->operator_list);
        if (k != NULL)
        {
            k->op->is_deleted = FALSE;
            k->op->x = node_x;
            k->op->name = node_x;
            k->op->met_x = node_x;
        }
        else
        {
            new_vertex = Make_Operator_Node("INPUT",
                "",    
                "INPUT",
                "",    
                GetUnique_Id(),
                0,     
                node_x,
                620,    
                2,      
                fake_color,
                node_x,
                640,
                fake_font,
                node_x,
                640,
                fake_font,
                node_x,
                640,
                FALSE,
                FALSE,
                FALSE,
                FALSE);
            Link_Operator(new_vertex, h);
            k = new_vertex;
        }/* add the new edges */
        edge_x = first_x;
        edge_label_y = 555;
        temp_p = h->inh_input_id_set;
        while (!BoolValue(IsNull(temp_p)))
        {
            edge_name = str0_to_str_ro(StrValue(Get_Id(FirstElement(temp_p))));
            temp_p = IDSetTail(temp_p);
            t = Edge_Name_Is_In_List(edge_name, NULL, k, h->stream_list);
            if (t != NULL)
            {
                t->is_deleted = FALSE;
                t->name_x = edge_x - 10;
                t->name_y = edge_label_y;
                t->arc->x = edge_x;
                t->arc->y = edge_y;
APPENDIX D - Auxiliary Functions

```c

if (output_count > 0)
{
    first_x = first_x + 80 * input_count + 100;
    node_x = (first_x + (80 * (output_count - 1) / 2));

    k = Vtx_Name_Is_In_List("OUTPUT", "OUTPUT", NULL, h->operator_list);
    if (k != NULL)
    {
        k->op->is_deleted = FALSE;
        k->op->x = node_x;
        k->op->name_x = node_x;
        k->op->met_x = node_x;
    } else {
        new_vertex = Make_Operator_Node("OUTPUT", "OUTPUT", NULL, Get_Undefined_Id(), 0, node_x, 620, 2, fake_color, fake_font, node_x, 640, fake_font, node_x, 640, FALSE, FALSE, FALSE, FALSE, FALSE);
        Link_Operator(new_vertex, h);
    }
    k = new_vertex;

    /* add the new edge */
    edge_x = first_x;
    edge_label_y = 555;
    temp_p = h->inh_output_id_set;
    while (!isNullStrNull(temp_p))
    {
        edge_name = str0_to_str_ro(StrValue(Get_Id(FirstElement(temp_p))));
        temp_p = IsSetTail(temp_p);

        t = Vtx_Name_Is_In_List(edge_name, k, NULL, h->stream_list);
        if (t != NULL)
        {
            t->is_deleted = FALSE;
            t->name_x = edge_x - 10;
            t->arc_x = edge_x;
            t->latency_x = edge_x - 10;
        }
    }
}
```
APPENDIX D - Auxiliary Functions

640,
FALSE,
FALSE,
FALSE,
FALSE,
FALSE,
FALSE;

Link_OPERATOR(new_vertex, h);
k = new_vertex;
}

/* add the new edges */
edge_x = first_x;
edge_label_y = 555;
temp_p = h->state_id_set;
while (!BoolValue((p)(temp_p)))
{
    edge_name = str0_to_str_ro((StrValue(Get_Id(FirstElement(temp_p)))));
temp_p = IdSetTail(temp_p);

    t = Edge_Name_Is_In_List(edge_name, k, k, h->stream_list);
    if (t != NULL)
    {
        t->is_deleted = FALSE;
        t->name_x = edge_x - 10;
        t->arc->x = edge_x - 5;
        t->arc->next->x = edge_x + 5;
        t->latency_x = edge_x - 10;
    }
    else
    {  
        spl_ptr = (SPLINE_PTR)malloc(sizeof(SPLINE_NODE));
        spl_ptr->next = (SPLINE_PTR)malloc(sizeof(SPLINE_NODE));

        spl_ptr->x = edge_x - 5;
        spl_ptr->y = 555;
        spl_ptr->next->x = edge_x + 5;
        spl_ptr->next->y = 555;
        spl_ptr->next = NULL;

        new_edge = Make_Stream_Node(edge_name,
                                    Get_Unique_Id(),
                                    k,
                                    fake_font,
                                    edge_x + 10,
                                    edge_label_y,
                                    fake_font,
                                    edge_x - 10,
                                    edge_label_y,
                                    SPL_PTR,
                                    false_latency,
                                    FALSE,
                                    FALSE,
                                    TRUE);
        Link_Stream(new_edge, h);
    }

    edge_x = edge_x + 80;
    edge_label_y = edge_label_y + 10;
}

void Remove.INPUT.Output.State.Nodes(h)
HeadPtr h;
OPnodePTR
Vtx_Name_Is_In_List(),
k;
STREAM
Edge_Name_In_List(),
t;

PRODINSTANCE

char
*edge_name;

if (h != NULL)
{
    k = Vtx_Name_Is_In_List("", "INPUT", ",", h->operator_list);
    if (k != NULL)
    {
        k->op->is_deleted = TRUE;
        temp_p = h->inh_input_id_set;
        while (!BoolValue((p)(temp_p)))
        {
            edge_name = str0_to_str_ro((StrValue(Get_Id(FirstElement(temp_p)))));
            temp_p = IdSetTail(temp_p);

            t = Edge_Name_Is_In_List(edge_name, NULL, k, h->stream_list);
            if (t != NULL)
            {
                t->is_deleted = TRUE;
            }
        }
    }
}

k = Vtx_Name_Is_In_List("", "OUTPUT", ",", h->operator_list);
if (k != NULL)
{
    k->op->is_deleted = TRUE;
    temp_p = h->inh_output_id_set;
    while (!BoolValue((p)(temp_p)))
    {
        edge_name = str0_to_str_ro((StrValue(Get_Id(FirstElement(temp_p)))));
        temp_p = IdSetTail(temp_p);

        t = Edge_Name_Is_In_List(edge_name, NULL, k, h->stream_list);
        if (t != NULL)
        {
            t->is_deleted = TRUE;
        }
    }

165
APPENDIX D - Auxiliary Functions

```c
k = Vtx_Name_Is_In_List("", "STATE", ",", h->operator_list);
if (k != NULL)
{
    k->op->is_deleted = TRUE;
    temp_p = h->state_id_ptr;
    while (!BoolValue(isNull(temp_p)))
    {
        edge_name = str_to_str_ro(StrValue(Get_Id(FirstElement(temp_p))));
        temp_p = IDSetTail(temp_p);
        t = Edge_Name_Is_In_List(edge_name, k, k, h->stream_list);
        if (t != NULL)
        {
            t->is_deleted = TRUE;
        }
    }
}
}

//-----------------------------------------------------------------------*/
void Remove_Dele ted_Operators(h)
    HeadPtr h;
    (OPNodePTR
        temp, k;
        k = h->operator_list;
        while (k != NULL)
        {
            if (k->op->is_deleted)
            {
                h->operator_list = k->next;
                free(k->op->name);
                free(k->op->optional_type_name);
                free(k->op->oper_name);
                free(k->op->parameter_list);
                free(k->op);
                free(k);
                k = h->operator_list;
            }
            else
            {
                k = NULL;
            }
        }
        if (h->operator_list != NULL)
        {
            k = h->operator_list;
            while (k->next != NULL)
            {
                if (k->next->op->is_deleted)
                {
                    temp = k->next;
                    k->next = temp->next;
                    free(temp->op->name);
                    free(temp->op->optional_type_name);
                    free(temp->op->oper_name);
                    free(temp->op->parameter_list);
                    free(temp->op);
                    free(temp);
                }
                else
                {
                    k = k->next;
                }
            }
        }
        */

void Remove_Deleted_Streams(h)
    HeadPtr h;
    (ST_PTR
        temp, t;
        int
            WipeOutSpline();
        t = h->stream_list;
        while (t != NULL)
        {
            if (t->st->is_deleted)
            {
                h->stream_list = t->next;
                WipeOutSpline(t->st->arc);
                free(t->st);
                free(t);
                t = h->stream_list;
            }
            else
            {
                t = NULL;
            }
        }
        if (h->stream_list != NULL)
        {
            t = h->stream_list;
            while (t->next != NULL)
            {
                if (t->next->st->is_deleted)
                {
                    temp = t->next;
                    t->next = temp->next;
                    WipeOutSpline(temp->st->arc);
                    free(temp->st);  
                    free(temp);
                }
                else
                {
                    t = t->next;
                }
            }
        }
```
void Restore_Deleted_Operators(h)
    HeadPtr h;
    { 
        OPENodePTR k;
        k = h->operator_list;
        while (k != NULL)
            { 
                k->op->is_deleted = false;
                k->op->is_modified = false;
                k->op->is_new = false;
                k = k->next;
            } 
    }

void Restore_Deleted_Streams(h)
    HeadPtr h;
    { 
        ST_PTR t;
        t = h->stream_list;
        while (t != NULL)
            { 
                t->st->is_deleted = false;
                t->st->is_modified = false;
                t->st->is_new = false;
                t = t->next;
            } 
    }

int Get_X(v)
    PROD_INSTANCE v;
    { 
        PROD_INSTANCE t,
        Get_Vertex_Time(),
        Convert_Time_To_Integer();
        t = Get_Vertex_Time(v);
        return(IntValue(Convert_Time_To_Integer(t))); 
    }

int Get_Latency(e)
    PROD_INSTANCE e;
    { 
        PROD_INSTANCE t,
        Get_Edge_Time(),
        Convert_Time_To_Integer();
        t = Get_Edge_Time(e);
        return(IntValue(Convert_Time_To_Integer(t))); 
    }

void Find_Type_Node_from_Data(p)
    PROD_INSTANCE p;
    { 
        char *name;
        TYPE_LIST temp_list;
        #ifdef SDE_DEBUG_1
        /* debugging */
        printf("Entering Find_Type_Node_from_Data\n");
        #endif
        name = str0_to_str_ro(StrValue(Get_Id(id_from_Data(p))));
        temp_list = Global_Type_List;
        while (temp_list != NULL)
            { 
                if (strcmp(name, temp_list->type_name) == 0)
                    return(temp_list);
                else
                    temp_list = temp_list->next;
            } 
        return(NULL);
    }

void Find_Header_Node_from_Op_or_TopImpl_or_TopSpec(p)
    PROD_INSTANCE p;
    { 
        int prod_no;
        HeadPtr h,
        Op_Number_Is_In_List(),
        Op_Name_Is_In_List();
        char *name;
        PROD_INSTANCE Get_Op_Name();
        #ifdef SDE_DEBUG_1
        /* debugging */
        printf("Entering Find_Header_Node_from_Op_or_TopImpl_or_TopSpec\n");
        #endif
        prod_no = (int)p;
        h = Op_Number_Is_In_List(prod_no);
        if (h == NULL)
            { 
                /* since op_id no corresponding to the Op production
                  instance may have changed since last update to the
                  op_node, need to check one more time for op_node with
                  matching name */
            } 
        }
APPENDIX D - Auxiliary Functions

```c
#ifdef SDE_DEBUG_2
    /* debugging */
    printf("Op_node id may have changed\n");
#endif

if (production(p) == prod_op)
    name = str_to_str_ro(StrValue(Get_Id(id_from_op(p))));
else if (production(p) == prod_t_op_impl)
    name = str_to_str_ro(StrValue(Get_Id(id_from_TopImpl(p))));
else
    name = str_to_str_ro(StrValue(Get_Id(id_from_TopSpec(p))));

if (name != NULL)
    h = Op_Name_Is_In_List(name);
if (h != NULL)
    /* update id of op_node */
    h->prod_no = prod_no;
)
return(h);

/*-----------------------------------------------*/
void Move_To_Structure_Front(p)
    HeadPtr p;
{
    extern HeadPtr
    prototype;
    HeadPtr
    Curr, Prev;

#ifdef GRAPHICS_DEBUG
    /* debugging */
    printf("Entering Move_To_Structure_Front\n");
#endif

    Prev = NULL;
    Curr = prototype;

    /* search for p and remove it from the "prototype" list if found */
    while (Curr != NULL)
    {
        if (Curr == p)
        {
            if (Prev == NULL)
            {
                prototype = Curr->next;
            } else
            {
                Prev->next = Curr->next;
            }
            Curr = NULL; /* to get out of the while loop */
        } else
        {
        }
    }
    /* reinsert p to front of the "prototype" list */
    p->next = prototype;
    prototype = p;

    /*-------------------------------*/
    PROD_INSTANCE Get_Id_From_Inputs_List(p)
    PROD_INSTANCE p;
{
    PROD_INSTANCE
    temp_p;
    Get_Id_From_Type_Decl();

#ifdef SDE_DEBUG_1
    /* debugging */
    printf("Entering Get_Id_From_Inputs_List\n");
#endif

    temp_p = p;
    while ((production(temp_p) != prod_input_list) &&
        (production(temp_p) != prod_inputs) &&
        (production(temp_p) != prod_type_decl))
    {
        temp_p = father(temp_p);
    }

    if (production(temp_p) == prod_input_list)
    {
#endif SDE_DEBUG_2
    /*debugging*/
    printf("Get_Id_From_Inputs_List; production(temp_p) == prod_input_list\n");
#endif

    /*
    temp_p = scn(temp_p, 1);
    /*
    temp_p = o_inputs_from_InputsListPair(temp_p);
    */
    if (production(temp_p) == prod_inputs)
    {
#endif SDE_DEBUG_2
    /*debugging*/
    printf("Get_Id_From_Inputs_List; production(temp_p) == prod_inputs\n");
#endif

    /*
    temp_p = scn(temp_p, 1);
    */
    temp_p = type_declarations_from_0pInputs(temp_p);
    */
    if (production(temp_p) == prod_type_decl)
    {
#endif SDE_DEBUG_2
    /*debugging*/
```

168
APPENDIX D - Auxiliary Functions

```c
printf("Get_Id_From_Inputs_List: production\(temp_p\) == prod_type_decl\n");
#endif

return\(Get_Id_From_Type_Decl\(temp_p\)\);
}
else
{
/* production\(temp_p\) == OpInputsNone */
return\(Make_IdNull\()\);
#ifdef SDL_DEBUG_1
/* debugging */
printf("Leaving Get_Id_From_Inputs_List\n");
#endif
/*---------------------------------------------*/
PROD_INSTANCE Get_Id_From_Inputs_List\(p\)
PROD_INSTANCE p;
{

PROD_INSTANCE

temp_p,
Get_Id_From_Type_Decl();

#ifdef SDL_DEBUG_1
/* debugging */
printf("Entering Get_Id_From_Inputs_List\n");
#endif

/* temp_p = p;*/
while ((production\(temp_p\) != prod_output_list) &&
   (production\(temp_p\) != prod_outputs) &&
   (production\(temp_p\) != prod_type_decl))
{
    temp_p = father\(temp_p\);
}

if (production\(temp_p\) == prod_output_list)
/* temp_p = son\(temp_p, 1\);*/
/* temp_p = o_outputs_from_InputsListPair\(temp_p\);*/
if (production\(temp_p\) == prod_outputs)
{
/* temp_p = son\(temp_p, 1\);*/
/* temp_p = type_declarations_from_OpOutputs\(temp_p\);*/
}
if (production\(temp_p\) == prod_type_decl)
{
return\(Get_Id_From_Type_Decl\(temp_p\)\);
}
else
{
/* production\(temp_p\) == OpInputsNone */
return\(Make_IdNull\());
}
#endif SDL_DEBUG_1
/* debugging */
printf("Leaving Get_Id_From_Inputs_List\n");
#endif
/*---------------------------------------------*/
PRODINSTANCE Get_Id_From_Stream\(p\)
PRODINSTANCE p;
{
PRODINSTANCE

temp_p,
Get_Id_From_Type_Decl();
#ifdef SDL_DEBUG_1
/* debugging */
printf("Entering Get_Id_From_Stream\n");
#endif

/* temp_p = p;*/
while ((production\(temp_p\) != prod_stream) &&
   (production\(temp_p\) != prod_type_decl))
{
    temp_p = father\(temp_p\);
}

if (production\(temp_p\) == prod_stream)
/* temp_p = son\(temp_p, 1\);*/
/* temp_p = type_declarations_from_Streams\(temp_p\);*/
return\(Get_Id_From_Type_Decl\(temp_p\)\);
#endif SDL_DEBUG_1
/* debugging */
printf("Leaving Get_Id_From_Stream\n");
#endif
/*---------------------------------------------*/
PRODINSTANCE Get_Operator_Id_From_CONSTRAINTS\(p\)
PRODINSTANCE p;
{
PRODINSTANCE

temp_p,
Make_IdNull();
#ifdef SDL_DEBUG_1
/* debugging */
printf("Entering Get_Operator_Id_From_CONSTRAINTS\n");
#endif

/* temp_p = p;*/
while ((production\(temp_p\) != prod_constraints) &&
   (production\(temp_p\) != prod_a_constraint))
{
    temp_p = father\(temp_p\);
}

if (production\(temp_p\) == prod_constraints)

/* temp_p = a_constraint_from_CONSTRAINTsPair\(temp_p\);*/
```
APPENDIX D - Auxiliary Functions

if (production[temp_p] == prod_a_constraint)
{
    return(operator_id_from_AConstraint(temp_p));
}
else
{
    return(OperatorIDNull);
}
#endif DEBUX_1
/* debugging */
print("Leaving Get_Operator_Id_From_Constraints\n");
#endif
/* --------------------------------------------------------------------------*/
PROC_INSTANCE Get_Id_From_Type_Decl(p)
PROC_INSTANCE p;
{
    PROC_INSTANCE
    temp_p,
    Make_IdNull(),
    IsTypeDeclNil(),
    IsDeclNil();
#endif DEBUX_1
/* debugging */
print("Entering Get_Id_From_Type_Decl\n");
#endif

if (production(temp_p) == prod_TypeDecl)
{
    return(Make_IdNull());
}
else
{
    temp_p = p;
    if (BoolValue(IsTypeDeclNil(temp_p)))
    {
        return(Make_IdNull());
    }
    else
    {
        /* temp_p = (temp_p, 1); */
        temp_p = a_decl_from_TypeDeclPair(temp_p);
    }
    if (BoolValue(IsADeclNil(temp_p)))
    {
        return(Make_IdNull());
    }
    else
    {
        /* temp_p = (temp_p, 1); */
        temp_p = id_list_from_ADecl(temp_p);
    }
    if (BoolValue(IsIdListNil(temp_p)))
    {
        return(Make_IdNull());
    }
    else
    {

    /*
    return(decl_from_TypeDeclPair(temp_p));
    */
    }
    if (production(op_id_pairs) == prod_operator_id_pairs)
    {
        op_id_pairs = Get_Parameters_From_Operator_Id_Pairs(op_id_pairs);
        if (production(op_id_pairs) == prod_operator_id_pairs)
        {
            return(id_from_IdPair(temp_p));
        }
    } //endif
#endif DEBUX_1
/* debugging */
print("Leaving Get_Id_From_Type_Decl\n");
#endif
/* --------------------------------------------------------------------------*/

char *Get_Parameters_From_Operator_Id_Pairs(op_id_pairs)
PROC_INSTANCE op_id_pairs;
{
    PROC_INSTANCE
    aid_list_1,
    aid_list_2;
    char
    *Get_Parameters_From_Alonen_Id_List(),
    *list_1,
    *list_2,
    dummy[100];
    if (production(op_id_pairs) == prod_operator_id_pairs)
    {
        op_id_pairs = Get_Parameters_From_Alonen_Id_List(aid_list_1);
        list_2 = Get_Parameters_From_Alonen_Id_List(aid_list_2);
        print("Get_Parameters before free list: dummy = \%s\n", dummy);
        /*
        free(list_1);
        free(list_2);
        */
        print("Get_Parameters after free list: dummy = \%s\n", dummy);
        /*
        return(strdup(dummy));
        */
        else
        {
            return(strdup("\"\"));
        }
    }
    /*
    Get_Parameters(before free list: dummy = \%s\n", dummy);
    */
#endif DEBUX_1
/* debugging */
print("Leaving Get_Id_From_Type_Decl\n");
#endif
/* --------------------------------------------------------------------------*/

char *Get_Parameters_From_Alonen_Id_List(aid)
PROC_INSTANCE aid;
{
    char
    *head_of_string,
    dummy[100];
    PROC_INSTANCE
    temp_p;
    temp_p = aid;
}
APPENDIX D - Auxiliary Functions

if (production(temp_p) != prod_alone_id_pair)
{
    return(strdup(""));
} else
{
    printf("dummy, "\n", "");

    temp_p = aidl;

    while (production(temp_p) == prod_alone_id_pair)
    {
        strcat(dummy, ","),
        strcat(dummy, str_to_str_ro(StrideValue(Get_Id(,
            id_from_AIdPair(temp_p))));
        temp_p = alone_id_list_from_AIdPair(temp_p);
    }

    /* print("Get_Parameters_From_AId_Pair: id_list = "\n", dummy);
    */
    /* skip the first comma */
    head_of_string = dummy;
    while (head_of_string != ",")
    {
        head_of_string ++;
    }

    return(strdup(head_of_string));
}

/*************************
 **** This fourth set of functions were already existing and
 **** for use with the Graphio Editor.
 **** Documented originally as file: ed3.srl
 *************************

if (OPNodePTR Vtx_Name_Is_In_List(optional_type_name, oper_name, parameter_list, p)
    OPNodePTR p,
    char 'optional_type_name,
    'oper_name,
    'parameter_list;

    /* Compare 'name' with the names stored in each of the operators in the
     list pointed to by 'p'.  If name is found, return a pointer to that
     node, otherwise, return NULL.
     */
    OPERATOR q;
    PROD_INSTANCE EqualOpId();
    
    ifdef SDE_DEBUG_3
    printf("Vtx_Name_Is_In_List: \n", \n", parameter_list);
    printf("Vtx_Name_Is_In_List: \n", \n", parameter_list);
    endif

    while (p != NULL)
    {
        q = p->op;
        ifdef SDE_DEBUG_3
        printf("Vtx_Name_Is_In_List: oper_name = \n", \n", q->oper_name);
        endif

        if (strcmp(q->name, "") == 0)
        {
            if (strcmp(oper_name, q->oper_name) == 0)
            {
                ifdef SDE_DEBUG_3
                printf("Vtx_Name_Is_In_List: found vertex\n");
                endif
                return(p);
            } else
            {
                ifdef SDE_DEBUG_3
                printf("Vtx_Name_Is_In_List: \n", \n", q->optional_type_name);
                printf("Vtx_Name_Is_In_List: parameter_list = \n", \n", q->parameter_list);
                endif
                if (strcmp(q->optional_type_name, \n", q->optional_type_name) == 0)
                    ifdef SDE_DEBUG_3
                    printf("Vtx_Name_Is_In_List: \n", \n", q->parameter_list);
                    endif
                    return(NULL);
                endif
                return(p);
            } else
            {
                p = p->next;
            } else
            {
                p = p->next;
            } else
            {
                p = p->next;
            } else
            {
                ifdef SDE_DEBUG_3
                printf("Vtx_Name_Is_In_List: can't find vertex\n");
                endif
                return(NULL);
            }
        }
    }
    
    /* debugging */
APPENDIX D - Auxiliary Functions

STREAM Edge_Name_Is_In_List(char *edge_name, from, to, p)
    char *from, *to;
    STREAM p;
    (STREAM q);
#endif GRAPHICS_DEBUG
    /* debugging */
    printf("Entering Edge_Name_Is_In_List\n");
#endif SDE_DEBUG_3
# ifdef GRAPHICS_DEBUG
    printf("Edge_Name_Is_In_List: edge_name = \%s\", edge_name);
    printf("Edge_Name_Is_In_List: from->name = \%s\", (from == NULL)?"EXTERNAL":from->oper_name);
    printf("Edge_Name_Is_In_List: from = \%d\", from);
    printf("Edge_Name_Is_In_List: to->name = \%s\", (to == NULL)?"EXTERNAL":to->oper_name);
    printf("Edge_Name_Is_In_List: to = \%d\", to);
#endif
    while (p != NULL)
        q = p->st;
#endif SDE_DEBUG_3
    printf("Edge_Name_Is_In_List: q->name = \%s\", q->name);
    printf("Edge_Name_Is_In_List: q->from->name = \%s\", (q->from == NULL)?"EXTERNAL":q->from->oper_name);
    printf("Edge_Name_Is_In_List: q->from = \%d\", (q->from == NULL)?"EXTERNAL":q->from);
    printf("Edge_Name_Is_In_List: q->to->name = \%s\", (q->to == NULL)?"EXTERNAL":q->to->oper_name);
    printf("Edge_Name_Is_In_List: q->to = \%d\", (q->to));
    #endif
    if (!strcmp(edge_name, q->name))
        {
            if (from == q->from && to == q->to)
                {
                    #ifdef SDE_DEBUG_3
                    printf("Edge_Name_Is_In_List: find edge\n");
                    #endif
                    return(q);
                } else {
                    p = p->next;
                } else {
                    p = p->next;
                }
        }
    else {
        return(NULL);
    }
#endif SDE_DEBUG_3
   return(NULL);
    }
#endif

HeadPtr Op_Name_Is_In_List(char *my_oper_name)
    char *my_oper_name;
    
    # ifdef GRAPHICS_DEBUG
    /* debugging */
    printf("Entering Op_Name_Is_In_List\n");
    #endif
    p = prototype;
    while (p != NULL)
        {
            if (strcmp(p->name, my_oper_name) == 0)
                return(p);
                else
                    p = p->next;
        }
    return(p);
#endif SDE_DEBUG_3
    HeadPtr Op_Number_Is_In_List(int prod_no)
    { extern HeadPtr prototype;
        HeadPtr p;
    # ifdef GRAPHICS_DEBUG
    /* debugging */
    printf("Entering Op_Number_Is_In_List\n");
    #endif
    p = prototype;
    while (p != NULL)
        {
            if (p->prod_no == prod_no)
                return(p);
            else
                p = p->next;
        }
    return(p);
}
APPENDIX D - Auxiliary Functions

fp = fopen(file_name, "w");
ops = psdl_components_from_Prot(Global_Prot);
while (production(ops) == prod_pedl_pair)
{
  temp_p = component_from_PedlPair(ops);
  if (production(temp_p) == prod_op)
  {
    #ifdef SDE_DEBUG_3
    printf("save_graphic_file: operator id = %s\n", str_to_str_ro(StrValue(Get_Id(id_from_op(temp_p)))));
    #endif
    p = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(temp_p);
    if (p != NULL)
    {
      fprintf(fp, "%s\n", "OPERATOR\n");
      fprintf(fp, "%s\n", p->name);
      Output_OperatorsS_Operators(fp, p->operator_list);
      Output_OperatorsS_Streams(fp, p->stream_list);
      fprintf(fp, "ENDOPERATOR\n");
    }
  }
  else if (production(temp_p) == prod_data)
  {
    if (production(type_impl_from_Data(temp_p)) == prod_type_impl)
    {
      temp_op_impl_list = operator_impl_list_from_TypeImpl(
        type_impl_from_Data(temp_p));
      while (production(temp_op_impl_list) == prod_op_impl_list_pair)
      {
        temp_t_op_impl = t_op_impl_from_OpImplListPair(temp_op_impl_list);
        if (production(temp_t_op_impl) == prod_t_op_impl)
        {
          #ifdef SDE_DEBUG_3
          printf("save_graphic_file: operator id = %s\n", str_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(temp_t_op_impl)))));
          #endif
          p = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(temp_t_op_impl);
          if (p != NULL)
          {
            fprintf(fp, "%s\n", "OPERATOR\n");
            fprintf(fp, "%s\n", p->name);
            Output_OperatorsS_Operators(fp, p->operator_list);
            Output_OperatorsS_Streams(fp, p->stream_list);
            fprintf(fp, "ENDOPERATOR\n");
          }
        }
      }
    }
  }
}

fclose(fp);
}
APPENDIX D - Auxiliary Functions

```c
#define SDE_DEBUG_3
printf("GRAPHIC INFORMATION SAVED\n");
#endif

void Output_Operators_Operators(fp, operator_list)
FILE
    *fp;
    OPNodePTR
    operator_list;
{
    OPERATOR
    q;
#endif SDE_DEBUG_1
/* debugging */
printf("Entering Output_Operators_Operators\n");
#endif

fprintf(fp, "OPERATORS\n");
while (operator_list != NULL)
{
    q = operator_list->op;
    if (!q->is_deleted)
    {
        fprintf(fp, "%s\n", q->name);
        fprintf(fp, "%d\n", q->x);
        fprintf(fp, "%d\n", q->y);
        fprintf(fp, "%d\n", q->radius);
        fprintf(fp, "%d\n", q->color);
        fprintf(fp, "%d\n", q->name_font);
        fprintf(fp, "%d\n", q->name_x);
        fprintf(fp, "%d\n", q->name_y);
        fprintf(fp, "%d\n", q->name_font);
        fprintf(fp, "%d\n", q->met_font);
        fprintf(fp, "%d\n", q->is_terminator==1?"TRUE":"FALSE");
    }
    operator_list = operator_list->next;
}

void Output_Operators_Streams(fp, stream_list)
FILE
    *fp;
    ST_PTR
    stream_list;
{
    STREAM
    q;
    SPLINE_PTR
    r;
#endif SDE_DEBUG_1
/* debugging */
printf("Entering Output_Operators_Streams\n");
#endif

fprintf(fp, "STREAMS\n");
while (stream_list != NULL)
{
    q = stream_list->st;
    if (!q->is_deleted)
    {
        fprintf(fp, "%s\n", q->name);
        fprintf(fp, "%d\n", (q->from == NULL?"NULL":q->from->op->name));
        fprintf(fp, "%d\n", (q->to == NULL?"NULL":q->to->op->name));
        if ((q->from != NULL) || (q->to != NULL))
        {
            fprintf(fp, "SPLINE\n");
            if (q->arc != NULL)
            {
                r = q->arc;
                while (r != NULL)
                {
                    fprintf(fp, "%d\n", r->x);
                    fprintf(fp, "%d\n", r->y);
                    r = r->next;
                }
            }
            fprintf(fp, "SPLINEEND\n");
        }
        fprintf(fp, "%d\n", q->name_font);
        fprintf(fp, "%d\n", q->name_x);
        fprintf(fp, "%d\n", q->name_y);
        fprintf(fp, "%d\n", q->latency_font);
        fprintf(fp, "%d\n", q->latency_x);
        fprintf(fp, "%d\n", q->latency_y);
        fprintf(fp, "%d\n", (q->is_state_variable==1?"TRUE":"FALSE");
    }
    stream_list = stream_list->next;
}

BOOL Process_File()
{
    BOOL
    result,
    Get_Operator_list(),
    Get_Stream_list();
    HeadPtr
    header,
    Get_Header();
    FILE
    *fopen(),
    *fp;
    int
    Print_Operators_Operators(),
    Print_Operators_Streams(),
```
APPENDIX D - Auxiliary Functions

Output_Structure();

char
buffer[30],
"file_name;"

ifdef SDE_DEBUG
/* debugging */
printf("Entering Process_File\n");
#endif

/* Load file name from where graphic attributes will be read */
fp = fopen("attr_file_name.grf", "r");
scanf(fp, "%s", buffer);

ifdef SDE_DEBUG
printf("%s\n", buffer);
#endif

file_name = (char *)malloc(strlen(buffer));
strcpy(file_name, buffer);

if (strcmp(file_name, "new") != 0)
{
fp = fopen(file_name, "r");
/* consume the "OPERATOR" word and continue or finish */
while ((n=fscanf(fp, "%s", buffer)) != EOF)
{
/* process an operator header */
header = Get_Header(fp);
if (header == NULL)
{
printf("Process_File: Get_Header FAILED\n");
close(fp);
return(FALSE);
}

/*process operator list*/
result = Get_Operator_List(fp, header);
if (result == FALSE)
{
printf("Process_File: Get_Operator_list FAILED\n");
close(fp);
return(FALSE);
}

/*process stream list */
result = Get_Stream_List(fp, header);
if (result == FALSE)
{
printf("Process_File: Get_Stream_List FAILED\n");
close(fp);
return(FALSE);
}

close(fp);
free(file_name);
/* the following command is commented out for ease of debugging 1/21/94 */
#endif Output_Structure(); /*
return(TRUE);
}
#endif

HeadPtr Get_Header(fp)
FILE *fp;

HeadPtr
Op_Name_Is_In_List(),
Make_Operator_Header(),
p;

void
Link_To_Structure();

char
buffer[30];

ifdef SDE_DEBUG
/* debugging */
printf("Entering Get_Header\n");
#endif

scanf(fp, "%s", buffer);

ifdef SDE_DEBUG
printf("%s\n", buffer);
#endif

p = Op_Name_Is_In_List(buffer);
if (p == NULL)
{
ifdef SDE_DEBUG
/* debugging*/
printf("Get_Header: header node with name %s not found, Aborted\n", buffer);
#endif

/* need to make new header node */
p = Make_Operator_Header(buffer, NULL, NULL, NULL, Get_Unique_Id(), 0, FALSE);
Link_To_Structure(p);

return(p);
}

BOOL Get_Operator_List(fp, head_node)
FILE *
fp;

HeadPtr
head_node;

[OPNodePTR
Make_Operator_Node(),

175
new_vertex;

char *
front_part,
*middle_part,
*suffix_part,
*get_vertex_type_name[],
*get_vertex_id_name[],
*get_vertex_parameters(),
*remove_blanks_from_string(),
*clean_name, 
buffer[100],
nname[100];

int
name_font,
name_x,
name_y,
met,
m_ned, 
x, 
y, 
color;

BOOL
is_terminator;

ONodePTR
n,
Vtx_Name_Is_In_List();

HeadPtr
k,
cp_Name_Is_In_List();

#ifndef SIZE_DEBUG
/* debugging */
print("Entering Get_OPERATOR_List\n");
#endif

/* eat away the OPERATORS word */
scanf(fp, "%*s", buffer);

while(fscanf(fp, "%*s", name) && (strcmp(name, 'STREAMS') != 0))
{
    fscanf(fp, "%d", &x);
    fscanf(fp, "%d", &y);
    fscanf(fp, "%d", &radius);
    fscanf(fp, "%f", &color);
    fscanf(fp, "%d", &xname_font);
    fscanf(fp, "%d", &xname_x);
    fscanf(fp, "%d", &xname_y);
    fscanf(fp, "%d", &xmet_font);
    fscanf(fp, "%d", &xmet_x);
    fscanf(fp, "%d", &xmet_y);
    fscanf(fp, "%s", buffer);
    is_terminator = (strcmp(buffer, "TRUE") == 0);
    clean_name = remove_blanks_from_string(name);
    front_part = get_vertex_type_name(clean_name);
    middle_part = get_vertex_id_name(clean_name);
    suffix_part = get_vertex_parameters(clean_name);

    n = Vtx_Name_Is_In_List(front_part, middle_part, suffix_part, head_node-> operator_list);
    if (n != NULL)
    {
        print("\Get_OPERATOR_List: duplicated vertex node with name %s\n", name); 
        return(FALSE);
    }
    else
    {
        /* need to create a new vertex node */
        new_vertex = Make_OPERATOR_Node(clean_name,
        front_part,
        middle_part,
        suffix_part,
        Get_Unique_Id(),
        NULL,
        x,
        y,
        radius,
        color,
        name_font,
        name_x,
        name_y,
        met_font,
        met_x,
        met_y,
        is_deleted,
        is_new,
        is_composite,
        is_terminator,
        is_modified);
        Link_OPERATOR(new_vertex, head_node);
    }
    free(clean_name);
    free(front_part);
    free(middle_part);
    free(suffix_part);
}

return(TRUE);

ifndef SIZE_DEBUG
/* debugging */
printf("Leaving Get_OPERATOR_List from node \n");
#endif
APPENDIX D - Auxiliary Functions

```c
from,
to,
Vtx_Op_Id_Is_In_List()

cp_list;

ST_PTR
Make_Stream_Node();
new_edge,
at;

STREAM
n,
Edge_Name_Is_In_List();

cchar
*optional_type_name,
*oper_name,
*parameter_list,
*get_vertex_type_name(),
*get_vertex_oper_name(),
*get_vertex_parameters(),
*remove_blanks_from_string(),
*clean_name,
buffer[40],
name[40];

SPLINE_PTR
GetSpline();

ST_PTR
sp;

BOOL
is_state_variable;

#ifdef SDE_DEBUG
/* debugging */
printf("Entering Get_Stream_List\n");
#endif

cp_list = head_node->operator_list;

while (fscanf(fp, "%s", name) && (strcmp(name, "ENDOPERATOR") != 0))
{
    #ifdef SDE_DEBUG
    printf("Get_Stream_List: stream name = %s\n", name);
    #endif

    fscanf(fp, "%s", buffer);
    clean_name = remove_blanks_from_string(buffer);
    if (strcmp(clean_name, "<NULL>") == 0)
    {
        #ifdef SDE_DEBUG
        printf("Get_Stream_List: to node = %s\n", clean_name);
        #endif
        to = NULL;
    }
    else
    {
        optional_type_name = get_vertex_type_name(clean_name);
        #ifdef SDE_DEBUG
        printf("Get_Stream_List: from node = %s\n", clean_name);
        printf("Get_Stream_List: optional_type_name = %s\n", optional_type_name);
        #endif
        oper_name = get_vertex_oper_name(clean_name);
        #ifdef SDE_DEBUG
        printf("Get_Stream_List: from node = %s\n", clean_name);
        printf("Get_Stream_List: oper_name = %s\n", oper_name);
        #endif
        parameter_list = get_vertex_parameters(clean_name);
        #ifdef SDE_DEBUG
        printf("Get_Stream_List: from node = %s\n", clean_name);
        printf("Get_Stream_List: parameter_list = %s\n", parameter_list);
        #endif

        from = Vtx_Name_Is_In_List(optional_type_name, oper_name,
                                    parameter_list, cp_list);

        free(clean_name);
        free(optional_type_name);
        free(oper_name);
        free(parameter_list);

        if (from == NULL)
        {
            printf("Get_Stream_List: from vertex %s not found\n",
                   clean_name);
            return (FALSE);
        }

        fscanf(fp, "%s", buffer);
        clean_name = remove_blanks_from_string(buffer);
        if (strcmp(clean_name, "<NULL>") == 0)
        {
            #ifdef SDE_DEBUG
            printf("Get_Stream_List: to node = %s\n", clean_name);
            #endif
            to = NULL;
        }
        else
        {
            optional_type_name = get_vertex_type_name(clean_name);
            #ifdef SDE_DEBUG
            printf("Get_Stream_List: to node = %s\n", clean_name);
            printf("Get_Stream_List: optional_type_name = %s\n", optional_type_name);
            #endif
            oper_name = get_vertex_oper_name(clean_name);
            #ifdef SDE_DEBUG
            printf("Get_Stream_List: to node = %s\n", clean_name);
            #endif
        }
    }
}
```

177
APPENDIX D - Auxiliary Functions

```c
printf("Get_Stream_List: oper_name = \%s\n", oper_name);
#endif
	parameter_list = get_vertex_parameters(clean_name);

#ifdef SDE_DEBUG
printf("Get_Stream_List: to node = \%s\n", clean_name);
printf("Get_Stream_List: parameter_list = \%s\n", parameter_list);
#endif

t = Vtx_Name_Is_In_List(optional_type_name, oper_name,
parameter_list, op_list);

free(clean_name);
free(optional_type_name);
free(oper_name);
free(parameter_list);
if (t == NULL)
{
    printf("Get_Stream_List: to vertex \%s not found\n", buffer);
    return(FALSE);
}

s = GetSpline(fp);

fscanf(fp, "%d", &name_font);
scanf(fp, "%d", &name_x);
scanf(fp, "%d", &name_y);
scanf(fp, "%d", &latency_font);
scanf(fp, "%d", &latency_x);
scanf(fp, "%d", &latency_y);

fscanf(fp, "%s", buffer);
if_state_variable = (strcmp(buffer, "TRUE") == 0);

n = Edge_Name_Is_In_List(name, from, to, head_node->stream_list);
if (n != NULL)
{
    printf("Get_Stream_List: duplicated stream with name \%s\n", name);
    return(FALSE);
}
else
{
    /* create new stream node */
    new_edge = Make_Stream_Node(name,
        GetUniqueId(),
        from,
        to,
        name_font,
        name_x,
        name_y,
        latency_font,
        latency_x,
        latency_y,
        t,
        fake_latency,
        fake_is_deleted,
        fake_is_new,
        fake_is_modified,
        fake_is_state
    );
    Link_Stream(new_edge, head_node);
    }
}
return(TRUE);

/* debugging */
printf("Entering GetSpline\n");
#endif

dummy = (SPLINE_PTR)malloc(sizeof(SPLINE_NODE));
head->next = NULL;
last = dummy;

/* eat away the SPLINE word */
while(fscanf(fp, "%s", buffer) && (strcmp(buffer, "SPLINEEND") != 0))
{
    scanf(fp, "%d", &x);
    scanf(fp, "%d", &y);
    p = (SPLINE_PTR)malloc(sizeof(SPLINE_NODE));
    p->x = x;
    p->y = y;
    p->next = NULL;
    last->next = p;
    last = p;
}
last = dummy->next;
free(dummy);
return(last);
}

/*---------------------------------------------*/
ST_PTR Make_Stream_Node(name, st_no, from, to,
    name_font, name_x, name_y,
    latency_font, latency_x, latency_y,
    spl, latency, is_deleted, is_new,
    is_modified, is_state)
{
    char
    *name;
    ONodePTR
    from,
    to;
    int
    name_font,
    name_x,
```
APPENDIX D - Auxiliary Functions

```c
name_y, latency_font, latency_x, latency_y, latency;

BOOL
in_deleted, in_new, in_modified, is_state;
SPLINE_PTR
spl;

ST_PTR
s;
STREAM
r;

#ifdef GRAPHICS_DEBUG
/* debugging */
printf("Entering Make_Stream_Node\n");
#endif

s = (ST_PTR)malloc(sizeof(ST_HEAD));
r = (STREAM)malloc(sizeof(ST_NODE));
r->name = strdup(name);
r->id = st_no;
r->from = from;
r->to = to;

/* r->from and r->to now points to the op_nodes which contain the info of the two end-points of the data stream */
r->arc = spl;
r->latency = latency;
r->is_state_variable = is_state;
r->is_new = is_new;
r->is_deleted = is_deleted;
r->is_modified = is_modified;

r->name_font = name_font;
r->name_x = name_x;
r->name_y = name_y;

r->latency_font = latency_font;
r->latency_x = latency_x;
r->latency_y = latency_y;

s->st = r;
s->next = NULL;
return(s);

*/

/*---------------------------*/
void Link_To_Structure(p)

HeadPtr p;
{
extern HeadPtr
prototype;

#ifdef SDE_DEBUG
/* debugging */
printf("Entering Link_To_Structure\n");
#endif

p->next = prototype;
prototype = p;

int Get_Unique_Id()

extern int
unique_id_count;

int temp;

temp = unique_id_count;
unique_id_count = unique_id_count + 1;
return(temp);

*/

Output_Structure()

int
PrintPrototypeS_Operators(),
PrintPrototypeS_Streams();

extern HeadPtr
prototype;

HeadPtr
p;

#ifdef SDE_DEBUG
/* debugging */
printf("Entering Output_Structure\n");
#endif

printf("---------- STRUCTURE ----------\n");
p = prototype;
while (p != NULL)
{
    printf("OPERATOR\n");
    printf("%s\n", p->name);
    printf("%d\n", p->op_id_no);
    printf("%s\n", (p->marked_for_delete == TRUE)?"TRUE":"FALSE");
    printf("%d\n", p->net);
    printf("%s\n", (p->is_composite==1)?"IS_COMPOSITE":"NOT_COMPOSITE");
    /* Print_Operators_Operators(p->op_list_list) */
    /* Print_Operators_Streams(p->stream_list); */
    printf("ENDOPERATOR\n");
    p = p->next;
}

*/

Print_Operators_Operators(c_list)

OPNodePTR c_list;
{ 
    OPERATOR
    q;

#ifdef SDE_DEBUG
/* debugging */
printf("Entering Print_Operators\n");
#endif

printf("OPERATORS\n");
while (c_list != NULL)
{
```
APPENDIX D - Auxiliary Functions

```c
q = o_list->op;

printf("o_list is \$dn\n", (int)o_list);
printf("o_list->op is \$dn\n", (int)q);
printf("name is \$dn\n", q->name);
printf("name_font is \$dn\n", q->name_font);
printf("op_id is \$dn\n", q->op_id);
printf("met is \$dn\n", q->met);
printf("met_font is \$dn\n", q->met_font);
printf("x is \$dn\n", q->x);
printf("y is \$dn\n", q->y);
printf("radius is \$dn\n", q->radius);
printf("color is \$dn\n", q->color);
printf("is_deleted is \$dn\n", q->is_deleted==1?"TRUE":"FALSE");
printf("new is \$dn\n", q->is_new==1?"TRUE":"FALSE");
printf("composite is \$dn\n", q->is_composite);
printf("terminator is \$dn\n", q->is_terminator==1?"TRUE":"FALSE");
printf("modified is \$dn\n", q->is_modified==1?"TRUE":"FALSE");
o_list = o_list->next;
}

/* Print_Operators_Stream() */
ST_PTR
s_list;

STREAM q;

SPLINE_PTR r;

#ifdef SDE_DEBUG_1
	/* debugging */
#endif

print("STREAMS
n");
while (s_list != NULL)
{
q = s_list->st;

printf("\$dn\n", q->name);
printf("\$dn\n", q->id);
printf("\$dn\n", q->from == NULL?"\$NULL\n": q->from->op->name);
printf("\$dn\n", q->to == NULL?"\$NULL\n": q->to->op->name);

printf("SPLINES
n");
r = q->arc;
while (r != NULL)
{
printf("\$dn\n", r->x);
printf("\$dn\n", r->y);
r = r->next;
}

printf("SPLINEEND
n");
printf("\$dn\n", r->latency);
printf("\$dn\n", q->is_deleted==1?"TRUE":"FALSE");
printf("\$dn\n", q->is_new==1?"TRUE":"FALSE");
printf("\$dn\n", q->is_state_variable==1?"TRUE":"FALSE");
printf("\$dn\n", q->is_modified==1?"TRUE":"FALSE");
}

s_list = s_list->next;
}

/*

This fifth set of functions were already existing and

designed to help enforce the consistency of PSDL program.

Documented originally as file: edl.sql

*/

/* this file contains all the routines needed to enforce consistency of the

PSDL program */

/*

The following definitions generate a Graphic Editor entry in the editor's
main menu. They are a faithful copy of what is stated in the synthesizer's
user's manual. Except that in the manual it is explained that a "c" file
with these definitions will do. While investigating this, I arrived to the
conclusion that such is not the case. The definition must be included in
an SSL file to work.

*/

Changes:
2/1/95 replacing all calls to the function Make_Declarations_Null
with calls to the function Make_Empty_Declarations

*/

#ifdef SDE_DEBUG_1
#endif

void Enforce_Constistency()
{
    PROD_INSTANCE
temp_p;
temp_son,
IDIsNull();

    void
Add_New_Ops_To_Proto(),
Clean_Op(),
Clean_Data();

#endif

print("Entering Enforce_Constancy\n");

Add_New_Ops_To_Proto();
temp_son = prod_pSDL_pair(temp_p);

/* temp_p now points to pSDL_components */
while (production(temp_p) == prod_pSDL_pair)
{
    temp_son = component_from_pSDLPair(temp_p);
    if (production(temp_son) == prod_op)
    {
#endif

print("Entering_Constancy: Op name = \$dn\n",
str0_to_str_ro(STRValue(Get_Op_Name(temp_son))));

180
APPENDIX D - Auxiliary Functions

```c
#include

if (IntValue(IdsNull(id_from_op[temp_non])) == 0)
    Clean_op[temp_non];
else if (production[temp_non] == prod_data)
{
    ifdef SDE_DEBUG_2
        printf("Enforce_Consistency: Type name = %s\n",
                str0_to_str_ro(Stride(Get_Id(id_from_data[temp_non]))));
    endif

    if (IntValue(IdsNull(id_from_data[temp_non])) == 0)
        Clean_Data[temp_non];

    /*
     * temp_p = son(temp_p, 2);
     * temp_p = pad1_components_from_padiPair(temp_p);
     */
    endif

    ifdef SDE_DEBUG_1
        printf("Leaving Enforce_Consistency\n");
    endif

    /*---------------------------------------------------------------*/

void Clean_op(p)
{
    Head PTR
    h,
    Find_Header_Node_from_op_or_TopImpl_or_TopSpec(),
    Make_Operator_Header();

    ATREE
    tempree,
    atree = bu_atree(br_buf(cur_browser));

    PROD_INSTANCE
    temp_p,
    new_op_spec,
    new_op_impl,
    new_inputs_list,
    new_outputs_list,
    new_net,
    new_stream_list,
    new_constraint_list,
    Get_Op_Name(),
    Make_Op(),
    Make_Inputs_List(),
    Make_Streams_List(),
    Build_Stream(),
    Make_StreamNull(),
    Build_Constraints(),
    Make_ConstraintsNull(),
    Op_Impl_Has_Null_Streams(),
    Op_Impl_Has_Null_Constraints(),
    Replace_Input_Output_Net(),
    Replace_Stream_Constraint_List(),
    Op_Impl_Is_Null(),
    Empty_Graph();
    /*
     Make_opimplNull();
     */
    void
    Print_Type_Decl();
    boolean
    need_new_op_spec,
    need_new_op_impl;

    ifdef SDE_DEBUG_1
        printf("Entering Clean_op\n");
    endif

    h = Find_Header_Node_from_op_or_TopImpl_or_TopSpec(p);
    if (h == NULL)
    {
    ifdef SDE_DEBUG_2
        //debugging"
        printf("clean_op: cannot find header_node\n");
    endif

        /* need to make new header node */
        h = Make_Operator_Header(str0_to_str_ro(Stride(Get_Op_Name(p))
        ),
        NULL, NULL, Get.Unique_Id(), (int)p, FALSE);
        Link.To_Structure(h);
    }

    ifdef SDE_DEBUG_2
        // debugging */
        printf("Clean_op: Op_Name = %s\n", h->name);
    endif

    if (!h->multi_op_error)
    {
        if (h->input_error)
        {
            /*
             Print_Type_Decl(h->inh_input_decl);
             */
            new_inputs_list = Make_Inputs_List(h->inh_input_decl);
            new_inputs_list = o_inputs_list_from_Operator_Spec(op_from_op[p]);
        }
        else
        {
            new_inputs_list = son(son(p, 2), 2);
            /*
             */
            if (h->output_error)
            {
```
Print_TypeDecl(h->inh_output_decl);
/*
 new_outputs_list = Make_Outputs_List(h->inh_output_decl);
 */
else
{

 new_outputs_list = scm(scm(p, 2), 3);
}
/*
 new_outputs_list = o_outputs_list_from_OperatorSpec(operator_spec_from_Op(p));
 */
if (h->error)
{

#endif
/*debugging*/
if (production(o_timing_info_from_OperatorSpec{
 operator_spec_from_Op(p)) == op_search("OpTimingInfo"))
{
 printf("Clean_Op: old_met = %d\n", IntValue(
 Convert_Time_To_Integer(
 time_from_OpTimingInfo(
 o_timing_info_from_OperatorSpec(
 operator_spec_from_Op(p))));
}
else
{
 printf("Clean_Op: old_met = 0\n");
}
printf("Clean_Op: new_met = %d\n", IntValue(
 Convert_Time_To_Integer(h->inh_met)));
#endif
new_met = Make_New_O_Timing_Info(h->inh_met,
 o_timing_info_from_OperatorSpec(operator_spec_from_Op(p));
}
else
{
 new_met = o_timing_info_from_OperatorSpec(operator_spec_from_Op(p));
}
if (h->input_error || h->output_error || h->error)
{
 h->input_error = false;
 h->output_error = false;
 h->error = false;
/*
 temp_p = scm(p, 2);
*/
 temp_p = operator_spec_from_Op(p);
 new_op_spec = Replace_Input_Output_Met(temp_p, new_inputs_list,
 new_outputs_list, new_met);
 need_new_op_spec = true;
}
else
{
 new_op_spec = operator_spec_from_Op(p);
 need_new_op_spec = false;
 }
/*
 (IntValue(Get_Impl_Form(operator_impl_from_Op(p)) > 0)
 & BoolValue(Empty_Graph(operator_impl_from_Op(p))))
{
 need_new_op_impl = true;
 new_op_impl = Make_OpImplNull();
 }
else
{
*/
 if (h->stream_error)
{
 new_stream_list = Build_Streams{
 IdSetDifference{
 h->edge_id_set,
 IdSetUnion{
 h->state_id_set,
 h->inh_input_id_set,
 h->inh_output_id_set)}};
 Global_TypeDecl;
}
else
{
 if (BoolValue(Op_Impl_Has_Non_Null_Streams(operator_impl_from_Op(p)))
{
 new_stream_list = optional_streams_from_Declarations{
 declarations_from_OperatorImpl{
 operator_impl_from_Op(p)));
 }
 else
 new_stream_list = Make_StreamNull();
 }
 if (h->constraint_error)
{
 if (BoolValue(Op_Impl_Has_Non_Null_Constraints(operator_impl_from_Op(p)))
{
 new_constraint_list = Build_Constraints(h->vertex_id_set,
 constraints_from_CC(
 cc_from_OperatorImpl{
 operator_impl_from_Op(p)));
 }
 else
 new_constraint_list = Build_Constraints{
 h->vertex_id_set, Make_ConstraintsNull();
 }
 else
{
 if (BoolValue(Op_Impl_Has_Non_Null_Constraints(operator_impl_from_Op(p)))
{
 new_constraint_list = constraints_from_CC(
 cc_from_OperatorImpl{
 operator_impl_from_Op(p)));
 }
APPENDIX D - Auxiliary Functions

}  
else
  new_constraint_list = Make_ConstraintsNull();
}

if (h->stream_error || h->constraint_error)
{
  h->stream_error = false;
  h->constraint_error = false;
  temp_p = operator_impl_from_Op(p);
  new_op_Impl = Replace_Streams_Constraint_List(temp_p,
      new_stream_list, new_constraint_list);
  need_new_op_impl = true;
}
else
{
  new_op_impl = operator_impl_from_Op(p);
  need_new_op_impl = false;
}
/*
*/

if (need_new_op_spec || need_new_op_impl)
{
  need_new_op_spec = false;
  need_new_op_impl = false;
  temp_tree = tree_to_atree(Make_OP(id_from_Op(p),
                     new_op_spec, new_op_impl));
  atree_is_not_maintained(temp_tree) = true;
  insert_placeholder_and_set_selection(atree, p);
  if (context(temp_tree) != context(atree))
  {
    insert_coersion(atree, temp_tree);
  }
  else
  {
    swap_selection(temp_tree, atree);
  }
  establish_rename_place(atree);
  rm_atree(temp_tree);
}
}
/*---------------------------------------------*/

void Clean_Data(p)
{
  PROD_INSTANCE p;
  {
    HeadPtr
    h,
    Find_Hdr(Node_from_Op_or_TopImpl_or_TopSpec(),
      Make_Op_Header());
    TYPE_LIST
    tl,
    Make_Type_Node(),
    Find_Type_Node_from_Data();
    ATREE
    temp_tree,
    atree = bu_atree(br_buf(cur_brightness));
    PROD_INSTANCE
    temp_p,
    temp_type_impl,
    temp_op_impl_list,
    temp_t_op_impl,
    new_op_impl,
    new_stream_list,
    new_constraint_list,
    Make_TopImpl(),
    Make_TopImplNull(),
    Build_Stream(),
    Make_StreamNull(),
    Build_Constraints(),
    Make_ConstraintsNull(),
    Op_Impl.Has_Null_Streams(),
    Op_Impl.Has_Null_Constraints(),
    Replace_Streams_Constraint_List(),
    Op_Impl.Is_Null(),
    Make_New_Op_Impl_List(),
    Concat_op_Impl_list(),
    Make_TypeImpl();
    /*
    */
    Make_TypeImplNull();
    /*
    */
    void
    Print_Type_Dcl();
    boolean
    need_new_op_impl;
    #ifdef SDR_DEBUG
    printf("Entering Clean_Data\n");
  endif

  tl = Find_Type_Node_from_Data(p);
  if (tl == NULL)
  { 
    tl = Make_Type_Node(str0_to_str_ro(setValue(id_from_Data(p))));
  }
  temp_type_impl = type_impl_from_Data(p);
  if (production(temp_type_impl) == prod_type_impl)
  {
    temp_op_impl_list =
      operator_impl_list_from_TypeImpl(temp_type_impl);
    while (production(temp_op_impl_list) == prod_op_impl_list_pair)
    {
      temp_t_op_impl = t_operator_impl_from_OpImplListPair(
          temp_op_impl_list);
      ...
APPENDIX D - Auxiliary Functions

```c
if (production(temp_t_op_impl) == prod_t_op_impl)
    {
        h = Find_Header_Node_from_Op_or_T0pImpl_or_T0pSpec(temp_t_op_impl);
        if (h == NULL)
            {
                #ifdef SDE_DEBUG_3
                    /* Debugging */
                    printf("Clean_Data: cannot find header_node\n");
                #endif
                /* need to make new header node */
                h = Make_Operator_Header(
                        str_to_str_ro( StrValue(Get_Id( id_from_T0pImpl(temp_t_op_impl)))),
                        NULL, NULL, Get_UInt32_Uid(), int32, FALSE,
                        Link_To_Structure(h));
            }
    }
    #ifdef SDE_DEBUG_3
        /* Debugging */
        printf("Clean_Data: Op_Name = %s\n", h->name);
    #endif
    if (BoolValue(IsElement(id_from_T0pImpl(temp_t_op_impl), t1->obsolete_op_impl))
            {
                temp_tree = tree_to_str(Make_T0pImplNull());
                atree_is_not_maintained(temp_tree) = true;
                insert_placeholder_and_set_selection(atree, temp_t_op_impl);
                if (context(temp_tree) != context(atree))
                    insert_coercion(atree, temp_tree);
                else
                    swap_selections(temp_tree, atree);
                establish_abstracted_place(atree);
                rm_atree(temp_tree);
            }
    else if (!!(h->multi_op_error))
        {
            /*
            if (BoolValue(Empty_Graph( operator_impl_from_T0pImpl(temp_t_op_impl))))
                { need_new_op_impl = true;
                  new_op_impl = Make_OpImplNull();
                }
            else
                {
                    /*
                    if (h->stream_error)
                        { new_stream_list = Build_Streams( IdSetDifference( h->edge_id_set, IdSetUnion( h->state_id_set, IdSetUnion( h->inh_input_id_set, h->inh_output_id_set))),
                          Global_Type_Decl1);
                    }
                    else
                        {
                            if (BoolValue(Op_Impl_Has_Non_Null_Stream( operator_impl_from_T0pImpl(temp_t_op_impl))))
                                { new_stream_list = optional_streams_from_Declarations( declarations_from_OperatorImpl( operator_impl_from_T0pImpl(temp_t_op_impl)));
                                    }
                                else
                                    new_stream_list = Make_StreamNull();
                                }
                    if (h->constraint_error)
                        { if (BoolValue(Op_Impl_Has_Non_Null_Constraints( operator_impl_from_T0pImpl(temp_t_op_impl))))
                            { new_constraint_list = Build_Constraints(h->vertex_id_set, constraints_from_CC( cc_from_OperatorImpl( operator_impl_from_T0pImpl(temp_t_op_impl))));
                                }
                            else
                                new_constraint_list = Build_Constraints( h->vertex_id_set, Make_ConstraintsNull());
                            }
                        else
                            { if (BoolValue(Op_Impl_Has_Non_Null_Constraints( operator_impl_from_T0pImpl(temp_t_op_impl))))
                                { new_constraint_list = constraints_from_CC( cc_from_OperatorImpl( operator_impl_from_T0pImpl(temp_t_op_impl)));
                                    }
                                else
                                    new_constraint_list = Make_ConstraintsNull();
                                }
                        }
                    if (h->stream_error || h->constraint_error)
                        { h->stream_error = false;
                          h->constraint_error = false;
                          temp_p = operator_impl_from_T0pImpl(temp_t_op_impl);
                          new_op_impl = Replace_Stream_Constraint_list(temp_p, new_stream_list, new_constraint_list);
                          need_new_op_impl = true;
                        }
                        else
                            {
```
APPENDIX D - Auxiliary Functions

new_op_impl = operator_impl_from_TcpImpl(temp_t_op_impl);
need_new_op_impl = false;
}

/*
 */

if (need_new_op_impl)
{
  need_new_op_impl = false;

temptree = tree_to_atree(Make_TcpImpl(
  id_from_TcpImpl(temp_t_op_impl),
  new_op_impl));
atree_is_not_maintained(temptree) = true;
insert_placeholder_and_set_selection(atree, temp_t_op_impl);
if (context(temptree) != context(atree))
{
  insert_coercion(atree, temptree);
}
else
{
  swap_selections(temptree, atree);
}
establish_resting_place(atree);
rm_atree(temptree);
}
}

/*
 */

establish_resting_place(atree);
rm_atree(temptree);
}

/*
 */

void Add_New_Ops_To Proto()
{
  PRODINSTANCE
  IsNull(),
  Make_Proto(),
  Merge_ладl_Components(),
  temp_op_set,
  temp_p,
  temp_new_p,
  component_p,
  Get_op_Name(),
  Make_New_Ops();

  ATREE
    temptree,
    atree = bu_atree(br_buf(cur_browser));

  char
  *component_name;

  boolean
  not_found;

  ifdef SDE_DEBUG
    printf("Entering Add_New_Ops_To Proto\n");
  endif

  temp_op_set = Global_Undef_Ops;
  if (!BoolValue(IsNull(temp_op_set))
  {
    temp_new_p = Merge_ладl_Components(
        Make_New_Ops(temp_op_set), ladl_components_fromProt(Global_Proto));
APPENDIX D - Auxiliary Functions

/*
 * Make_New_Ops(temp_op_set), ssn(GlobalProto, 1));
 */

move_selection(atree, one_point_selection(GlobalProto));
tempree = tree_to_atree(Make_Prototemp_new_p);
if (context(tempree) != context(atree))
{
#define SDR_DEBUG_3
printf("Add_New_Ops: insert_coercion\n");
#undef SDR_DEBUG_3
    insert_coercion(atree, tempree);
} else
{
#define SDR_DEBUG_3
printf("Add_New_Ops: swap_selection\n");
#undef SDR_DEBUG_3
    swap_selections(tempree, atree);
    establish_resting_place(atree);
    rm_atree(tempree);
    br_set_insert_pt_to_selection(cuor_browser);
    cmd_cond_modifies(cuor_browser, cur_buffer);
    br_paint_all();
}

foreign Build_inputsList(i, o, td)

prod_instance
i,/* id */
o;/psdi_components consisting of all operators */
td;/all type declarations */
{
prod_instance
temp_input_ids,
temp_type_decl,
Extract_input_id_set(),
IsNull(),
Make_inputsListPair_from_type_decl(),
Make_inputsListNone();

temp_input_ids = Extract_input_id_set(i, o);
if (!BoolValue(IsNull(temp_input_ids))
{
temp_type_decl = Build_Type_Dec(temp_input_ids, td);
return(Make_inputsListPair_from_type_decl(temp_type_decl));
} else
{
return(Make_inputsListNone());
}
}

foreign Build_outputList(i, o, td)

prod_instance
i,/* id */
o;/psdi_components consisting of all operators */
td;/all type declarations */
{
prod_instance
temp_input_ids,
temp_type_decl,
Extract_output_id_set(),
IsNull(),
Make_outputListPair_from_type_decl(),
Make_outputListNone();

temp_input_ids = Extract_output_id_set(i, o);
if (!BoolValue(IsNull(temp_input_ids))
{
temp_type_decl = Build_Type_Dec(temp_input_ids, td);
return(Make_outputListPair_from_type_decl(temp_type_decl));
} else
{
return(Make_outputListNone());
}

void Free_linked_list(l_l)

linked_list l_l;
{
linked_list head;
while (l_l != NULL)
{
    head = l_l;
    l_l = l_l->next;
    free(head);
}
}

void Print_type_decl(td)

prod_instance td;
{
prod_instance
temp_p,
Get_id(),
Get_id_from_type_decl();
temp_p = td;
while (production(temp_p) == prod_type_decl)
{
    printf("Print_type_decl: Id = %s\n",  
        str0_to_str_ro(STRValue(Get_id(Get_id_from_type_decl(temp_p)))));
    temp_p = type_declarations_from_type_decl_pair(temp_p);
}
}

FOREIGN Build_outputList(i, o, td)
# APPENDIX D - Auxiliary Functions

```c
void Add_Root_Op(name)
    char *name;
    {
        ATREE atree = bu_atree(br_buf(cur_browser));
        ATREE tempree;
        PROD_INSTANCE
        Make_Of_From_SSLetring(),
        temp_p,
        temp_new_p;
        PRODUCTION
        p_comp_prod = op_search("PsdlPair");

    #ifdef SDE_DEBUG_1
        printf("Entering Add_Root_Op\n");
    #endif

    temp_p = psdl_components_from_Prot(Global_Proto);

    insert_placeholder_and_set_selection(atree, temp_p);
    temp_new_p = Make_Of_From_SSLetring(SSSString(name));
    tempree = tree_to_atree(temp_new_p);
    atree_is_not_maintained(tempree) = true;

    if (context(tempree) != context(atree))
        insert_coercion(atree, tempree);
    else
        swap_selections(tempree, atree);

    establish_resting_place(atree);
    rm_atree(tempree);

    /* update atree, buffer and selection */
    br_set_insert_pt_to_selection(cur_browser);
    cmd_cond_modifies(cur_browser, cur_buffer);
    br_paint_all();

    #ifdef SDE_DEBUG_1
        printf("Leaving Add_Root_Op\n");
    #endif

    } /*-----------------------------------------------*/

void sort_psdl_components()
    {
        ATREE tempree,
        atree = bu_atree(br_buf(cur_browser));
        PROD_INSTANCE

        LINKED_LIST
        temp_head,
        current_pos_trace = NULL;

        void
        Free_Linked_List();

    #ifdef SDE_DEBUG_1
        printf("Entering sort_psdl_components\n");
    #endif

    /* remember the cursor position */
    p = selection_psm(SE_selection(br_atree(br_buf(cur_browser))));
    temp_p = p;
    current_pos_trace = NULL;

    while (!at_top(temp_p) && (production(temp_p) != prod_op)
        && (production(temp_p) != prod_data))
    {
        if (prod_no_component) ;
        printf("sort_psdl_components: top_id = %d\n", prod_no_component);
        
        #ifdef SDE_DEBUG_3
            printf("sort_psdl_components: top_id = %d\n", prod_no_component);
        #endif

        if (!at_top(temp_p) && (production(temp_p) != prod_no_component))
            {
                top_id = id_from_Op_or_Data(temp_p);
            }
        else
            {
                top_id = Make_IdNull();
            }

        #ifdef SDE_DEBUG_3
            printf("sort_psdl_components: top_id = %d\n", prod_no_component);
        #endif

        move_selection(atree, one_point_selection(Global_Proto));
        tempree = tree_to_atree(Make_Proto(Sort_Psdl_Components{
                psdl_components_from_Prot(Global_Proto)}));
        atree_is_not_maintained(tempree);
```
if (context(temp_tree) != context(tree))
    insert coercion(tree, temp_tree);
else
    swap_selections(temp_tree, tree);

establish_renaming_place(tree);
rm_atree(temp_tree);
br_set_insert_pt_to_selection(cur_browser);
cmd_cond_modifies(cur_browser, cur_buffer);
br_paint_all();

#define SDE_DEBUG_3
    printf("sort_padi_components: finished swapping trees\n");
#endif

/* move cursor back to original position */
if (IntValue(DisNull(top_id)) == 0)
{
    temp_p = Find_Component(top_id, padi_components_from_form(Global_Protos));
    temp_head = current_pose_trace;
    while (temp_head != NULL)
    {
        temp_p = scn(temp_p, (temp_head->item_number));
        temp_head = temp_head->next;
    }
    p = temp_p;
    Free_Linked_List(current_pose_trace);
    temp_p = one_point_selection(p);
    move_selection_atree(temp, temp_p);
br_set_insert_pt_to_selection(cur_browser);
cmd_cond_modifies(cur_browser, cur_buffer);
br_paint_all();
}
#endif SDE_DEBUG_1
    printf("Entering edit_graph\n");
#endif

/*-----------------------------------------------*/

void edit_graph()
{
ATREE
temp_tree,
    tree = bu_atree(br_buf(cur_browser));

PROD_INSTANCE
    junk_p;

PROD_INSTANCE
    lex_element(),
    new_graph,
    Edit_Graph(),
    Make_Op_Impl(),
    Make_CCNull(),
    /*
    Make DeclarationsNull(),
    */
    Make_Empty_Declarations(),
    Make_Id_from_SSLString(),
    Get_Impl_Form(),
    Find_Component(),
    Find_Op_Impl_ID_Data(),
    current_op_id,
    component_id,
    op_impl_p,
    temp_p;

PRODUCTION
top_production;

HeadPtr
    h,
    parent_head_ptr,
    Find_Parent_Name(),
    Find_Header_Name_from_Op_or_T0pImpl_or_T0pSpec();

char
    *vertex_type_name,
    *component_name;

void
    House_Cleaning(),
    Refresh_Graph_Viewer();

#define SDE_DEBUG_1
    printf("Leaving sort_padi_components\n");
#endif

while (!at_top(temp_p) && (production(temp_p) != prod_op) && (production(temp_p) != prod_t_op_impl))
{
    temp_p = selection_apex(SE_selection(br_atree(br_buf(cur_browser))));
    if (at_top(temp_p))
    {
        write_error_string("must select an operator before invoking edit-graph");
        br_paint_all();
    }
    else
    {
        if (production(temp_p) == prod_op)
        {
            current_op_id = Id_from_Op(temp_p);
            op_impl_p = operator_impl_from_op(temp_p);
            top_production = prod_op;
        }
        else
        {
            current_op_id = Id_from_T0pImpl(temp_p);
            op_impl_p = operator_impl_from_T0pImpl(temp_p);
            top_production = prod_t0p_impl;
        }
    }
}
APPENDIX D - Auxiliary Functions

if (IntValue(IdIsNull(current_op_id)) != 0)
    { write_error_string("need to give operator name before invoking edit-graph");
        br_paint_all();
    } else
    {
        Global_Refresh_Graph_Visual = true;
        // initialize the global variables error_message and ge_result */
        if (error_message != NULL)
            free(error_message);
        ge_result = NO_UPDATES;
        new_graph = Edit_Graph();
        if (ge_result == GO_UP || ge_result == SAME_LEVEL || ge_result == GO_DOWN)
        {
            if (IntValue(Get_Impl_Form(op_impl_p)) == 0)
                { #ifdef SDR_DEBUG_3
                    printf("edit_graph: make_Impl
                    temp_tree = tree_to_astree(
                        Make_Impl(new_graph, Make_DeclarationsNull()),
                        Make_Impl(new_graph, Make_Empty_Declarations(), Make_CtNull()));
                } else
                { #ifdef SDR_DEBUG_3
                    printf("edit_graph: new_graph
                    temp_tree = tree_to_astree(
                        Make_Impl(new_graph, declarations_from_OperatorImpl(op_impl_p),
                        cc_from_OperatorImpl(op_impl_p)));
                    astree_is_not_maintained(temp_tree);
                    #ifdef SDR_DEBUG_3
                        printf("after tree surgery
                    if (context[temp_tree] != context(atre))
                        insert_coercion(atre, temp_tree);
                    else
                        insert_selections(temp_tree, atre);
                } #ifdef SDR_DEBUG_3
            printf("after tree surgery
        #endif
       Ƭ
        st_tree[temptree];
        if (context[st_tree] != context(atre))
            insert_coercion(atre, st_tree);
        else
            insert_selections(st_tree, atre);
            if (str cmp(h->parent, "") == 0)
                { #ifdef SDR_DEBUG_3
                    printf("after tree surgery
                    parent_head_ptr = Find_Parent_Name(h->name);
                } #ifdef SDR_DEBUG_3
            printf("after tree surgery
        #endif
        parent_type_name = strdup(parent_head_ptr->type_id);
        h->parent_type_name = strdup(h->parent_type_name);
        h->parent_type_name = strdup(h->parent_type_name);
            if (str cmp(h->parent_type_name, "") == 0)
                component_name = h->parent_type_name;
            else
                component_name = h->parent;
        
        h->parent = strdup(parent_head_ptr->name);
        h->parent_type_name = strdup(h->parent_type_name);
        
        if (str cmp(h->parent_type_name, "") == 0)
            component_name = h->parent_type_name;
        else
            component_name = h->parent;
APPENDIX D - Auxiliary Functions

#define SDE_DEBUG_3
printf("edit_graph: parent_component_name = \$s\$", component_name);
#else
#endif

if (ge_result == GO_DOWN)
{
    if (strcmp(goto_child, "") == 0)
    {
        write_error_string("must select a labelled operator to be decomposed before exiting graph editor");
    }
    else
    {
        vertex_type_name = get_vertex_type_name(goto_child);
        if (strcmp(vertex_type_name, "") != 0)
        {
            write_error_string("can't decompose type-operator within an operator impl");
        }
    }
    component_name = "";
}
else
{
    component_name = get_vertex_oper_name(goto_child);
}
else
{
    component_name = "";
}
#endif

if (strcmp(component_name, "") != 0)
{
    printf("edit_graph: component_name =\$s\$", component_name);
    component_id = Make_Id_From_SSSLstring(SSSstring(component_name));
    component_id = Id(SSSstring(component_name));
    temp_p = Find_Component(component_id,
                        pail_components_from_prot(Globa_prot));
    if (production(temp_p) != prod_no_component)
    {
        if (production(temp_p) == prod_data)
        {
            temp_p = type_impl_from_data(temp_p);
        }
        else
        {
            write_error_string("parent has empty type impl");
        }
    }

    temp_p = FindTopImpl_in_operator_impl_list(

    if (strncmp(temp_p, "", 1) == 0)
    {
        printf("Leaving edit_graph\n");
    }
#endif

/* --------------------------------------------------------------------------*/
HeadPtr Find_Parent_Name(child_name)
(
    char *child_name;
    boolean not_found = true;
    HeadPtr h = prototype;
    OPNodePtr p_list;
    /*
APPENDIX D - Auxiliary Functions

```c
char
*parent_name;
*
#endif
#define SDE_DEBUG_1
print("Entering Find_Parent_Name\n");
#endif
#endif SDE_DEBUG_3
print("Finding Parent_Name: child_name = \%s\n", child_name);
#endif
while (!not_found && h != NULL)
{ p_list = h->operator_list;
    while (!not_found && p_list != NULL)
    { #ifdef SDE_DEBUG_3
      print("Finding Parent_Name: target_name = \%s\n", p_list->op->name);
      #endif
        if (strcmp(child_name, p_list->op->name) == 0)
          { not_found = false;
            /* parent_name = h->name; */
          }
        else
          p_list = p_list->next;
    }
    if (not_found)
      h = h->next;
}
if (not_found)
/*

*/
#endif SDE_DEBUG_3
print("Finding Parent_Name: not found\n");
#endif
return(NULL);
} else
# ifdef SDE_DEBUG_3
print("Finding Parent_Name: found parent\n");
#endif
print("Finding Parent_Name: h->name = \%s\n", h->name);
#endif
return(h);
#endif SDE_DEBUG_1
print("Leaving Find_Parent_Name\n");
#endif

FOREIGN Get_Ada_Op_Impl()
{

PROD_INSTANCE
temp_p,
Make_AdaOpImpl();

temp_p = selection_apex(SE_selection(bu_stree(br_buf(cur_browser))));
while (!at_top(temp_p) && (production(temp_p) != prod_op) &&
   (production(temp_p) != prod_t_op_impl))
    { temp_p = father(temp_p); }
}
if (!at_top(temp_p))
{ write_error_string("must select an operator before invoking transformation");
  br_paint_all();
  return(NULL); }
else
if (!production(temp_p) == prod_op)
  return(Make_AdaOpImpl(id_from_op(temp_p)));
else
  return(Make_AdaOpImpl(id_from_TopImpl(temp_p))); 
}

/*----------------------------------------*/
FOREIGN Remove_Leading_Blanks_From_String(s)

PROD_INSTANCE(s);{
char	*temp_string;
int i;
temp_string = str0_to_str1r(Salir_value(s));
#endif SDE_DEBUG_3
/*debugging*/
print("Remove_Leading_Blanks_From_String: temp_string = \%s\n", temp_string);
#endif
while ((*temp_string) != NULL && (((*temp_string) == ' ') ||
                  ((*temp_string) == ' '))
    temp_string ++;
return(CommentLine(SSSLstring(temp_string))); 
}

/*----------------------------------------*/
FOREIGN Get_Ada_Type_Impl()
{

PROD_INSTANCE
temp_p,
Make_AdaTypeImpl();

temp_p = selection_apex(SE_selection(bu_stree(br_buf(cur_browser))));

191
while (at_top(temp_p) && (production(temp_p) == prod_data))
{
  temp_p = father(temp_p);
}

if (at_top(temp_p))
{
  write_error_string("must select an operator before invoking transformation");
  br_paint_all();
  return(NULLVALUE);
}
else
{
  /*
   return(Make_AdsTypeImpl{mon{temp_p, 1}});
   */
  return(Make_AdsTypeImpl{id_from_Data{temp_p}});
}

FOREIGN Build_Met(i, o)
PROD_INSTANCE
i; /* operator_id */
o; /* psdl_components consisting of all operators */
{
  PROD_INSTANCE
  temp_met,
  Extract_Met{()},
  Valid_Met{();
  temp_met = Extract_Met{i, o};
  if (BoolValue(Valid_Met{temp_met}))
  {
    return(Make_OpTimingInfo_From_Met{temp_met});
  }
  else
  {
    return(Make_OpTimingInfoNone{());
  }
}

void Remove_empty_graph()
{
  PROD_INSTANCE
  p;
  Empty_Graph{(),
  Make_Op{(),
  Make_OpImplNull{()},
  temp_op_impl_list,
  temp_t_op_impl,
  temp_p,
  temp_mon,
  ATREE
  temp_tree,
  atree = bu_atree(br_buf(cur_browser));
  p = psdl_components_from_Prot(Global_Protos);
  while (production(p) == prod_psdl_pair)
  {
    temp_mon = component_from_PsdlPair{p};
    if (production(temp_mon) == prod_op)
    {
      temp_p = operator_impl_from_Op{temp_mon};
      if ((production(temp_p) == prod_op_impl) && BoolValue(Empty_Graph{temp_p}))
      {
        temptree = tree_to_atree(Make_Op{id_from_op{temp_mon},
          operator_spec_from_op{temp_mon},
          Make_OpImplNull{()});
        atree_is_not_maintained(temptree) = true;
        insert_placeholder_and_set_selection(atreer, temp_mon);
        if (context(temptree) != context(atreer))
        {
          insert_coersion(atreer, temp_mon);
        }
        else
        {
          swap_selections(temptree, atree);
        }
        establish_resting_place(atreer);
        rm_atree(temptree);
      }
      else
      {
        if (production(type_impl_from_Data{temp_mon}) == prod_type_impl)
        {
          temp_op_impl_list = operator_impl_list_from_TypeImpl{
            type_impl_from_Data{temp_mon}};
          while (production(temp_op_impl_list) == prod_op_impl_list_pair)
          {
            temp_t_op_impl = t_oper_impl_from_OpImplListPair{
              temp_op_impl_list};
            if (production(temp_t_op_impl) == prod_t_op_impl)
            {
              temp_p = operator_impl_from_TpImpl{temp_t_op_impl};
              if ((production(temp_p) == prod_op_impl) && BoolValue(Empty_Graph{temp_p}))
              {
                temptree = tree_to_atree(Make_TpImpl{
                  id_fromTpImpl{temp_t_op_impl},
                  Make_OpImplNull{()});
                atree_is_not_maintained(temptree) = true;
                insert_placeholder_and_set_selection(atreer, temp_pimpl);
                if (context(temptree) != context(atreer))
                {
                  insert_coersion(atreer, temp_pimpl);
                }
                else
                {
                  swap_selections(temptree, atree);
                }
              }
            }
          }
        }
      }
  }}
establish_resting_place(atree);
    rm_stree(tempstree);
    
    temp_op_impl_list = operator_impl_from_TOpImpl(
        temp_op_impl_list);
    
    p = psdl_components_from_PsdlPair(p);
}

/*----------------------------------------------------------------------------*/

/**
   * This sixth set of functions were already existing and was
   * documented originally as file: ed10.sx1
   *----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------/
   * FOREIGN FUNCTION DEFINITIONS -------------------------------------------*/

/*--------------------------- graph foreign Edit_Graph(); */

/*--------------------------- o_inputslist foreign Build_InputsList(id i, psdl_components o, type_declarations tld);
   o_outputlist foreign Build_OutputsList(id i, psdl_components o, type_declarations tld);
   o_timinginfo foreign Build_TimingInfo(operator_id i, psdl_components o);
   BOOL foreign Is_Show_Graph_Text_View();
   operator_impl foreign Get_Ada_Op_Impl();
   type_impl foreign Get_Ada_Type_Impl();
   t_op spec foreign Find_T_Op_Spec_In_Data(id o id, id t id);
   commentLine foreign Remove_Ledding_Blanks_From_String(CLINE i); */

/*--------------------------- */

_EOF_SOURCE_FILE_-

*/

/** PROCEDURE call_graph_editor() */

ifdef SDE_DEBUG
  /* debugging */
  printf("Entering Call_graph_editor\n");
endif

/* the following if statement is added to catch that case
   in which the user invokes the graph editor without
   selecting the structure to be edited. */

if (current_graph != NULL) {
  edit();
}

ifdef SDE_DEBUG2
  /* debugging */
  printf("Back from GE\n");
endif
else {
  write_error_string("must select an operator before invoking graph editor");
  /*
    printf("Error: you need to select the structure to be edited\n");
    printf(" before invoking the graphic editor\n");
    */
}

PROCEDURE save_graphic_attributes()
{
    clean_buffer_no_reclaim();
    extern
APPENDIX D - Auxiliary Functions

foreign Is_Show_Graph_Text_View()
{
    extern VIEW_NO cur_view;
    return(
        (cur_view == name_to_view("SHOW_GRAPH_TEXT_VIEW")
            ? SQL_true
            : SQL_false
        )
    );
}

foreign Edit_Graph()
{
    PROD_INSTANCE
    op_id,
    IdIsNull(),
    Make_Graph(),
    old_graph,
    temp_p,
    boolean
    different();
    OPModePTR
    o_list,
    Sort_Operator_List();
    ST_PTR
    n_list,
    Sort_Stream_List();
    HeadPtr
    h,
    Find_Header_Node_from_Op_or_TopImpl_or_TopSpec();
    Make_Operator_Header();
    void
    Remove_Components();
    Remove_Deleted_Operators();
    Restore_Deleted_Operators();
    Restore_Deleted_Streams();
    Add_Output_Stream();
    Remove_Input_Stream();
    Update_Operator();
    Move_To_Structure();
    Link_To_Structure();

    temp_p = selection_spec(SE_selection(bu_stree(br_bu[(cur_browser)])));
    while (!list_top(temp_p)
        && production(temp_p) != prod_top
        && production(temp_p) != prod_t_op_impl)
        temp_p = father[temp_p];
    if (at_top(temp_p)
        write_error_string("Need to select an operator before edit graph");
        return(Make_Graph(NULL, NULL));
    } else
        if (production[temp_p] == prod_op
            op_id = id_from_op(temp_p);
        else
APPENDIX D - Auxiliary Functions

```c
op_id = id_from_TopImpl(temp_p);

if (!IntValue(IdsIsNull(op_id))) {
    write_error_string("need to give operator name before edit graph");
    return(Make_Graph(NULL, NULL));
} else {
    h = Find_Header_Node_from_Op_or_TopImpl_or_TopSpec(temp_p);
    if (h == NULL) {
        /* need to make new header node */
        h = Make_Graph_Header(str0_to_str_to(StringValue(Get_Id(op_id))),
                               NULL, NULL, Get_Use_Name(),
                               (int)temp_p, FALSE);
        Link_To_Structure(h);
    } else {
        Move_To_Structure_Front(h);
    }

    Update_Operators(temp_p, h);
    Add_Input_Output_State_Nodes(h);
    current_graph = h;
    the_operator_list = h->operator_list;
    the_stream_list = h->stream_list;

    /* current_graph cannot be NULL at this point */

    #ifdef SDE_DEBUG_3
    print("------- Edit_Graph: before call_graphic_editor -------\n");
    Print_Operators_Operators(current_graph->operator_list);
    Print_Operators_Operators(current_graph->stream_list);
    #endif

    call_graphic_editor();

    #ifdef SDE_DEBUG_3
    print("------- Edit_Graph: after call_graphic_editor -------\n");
    Print_Operators_Operators(current_graph->operator_list);
    Print_Operators_Operators(current_graph->stream_list);
    #endif

    Remove_Input_Output_State_Nodes(h);

    if (ge_result != NO_UPDATE && ge_result != ERROR) {
        /* current_graph is a global variable which will be modified by
call_graphic_editor */
    }
    Remove_DELETED_Operators(h);
    Remove_DELETED_Streams(h);
}

o_list = Sort_Operator_List(current_graph->operator_list);
        current_graph->operator_list = o_list;
    s_list = Sort_Operator_List(current_graph->stream_list);
        current_graph->stream_list = s_list;
    temp_p = Make_Graph(o_list, s_list);
    return(temp_p);
}
```

```c
boolean different(p, q)
    PROD_INSTANCE p, q;
    {
        register int i;
        register PRODUCTION prodp, prodq;
        COMPARISON test;

        if (p == q)
            return(FALSE);
        if (p == NULLVALUE || q == NULLVALUE)
            return(TRUE);

        prodp = production(p);
        prodq = production(q);
        if (prodp != prodq)
            return(TRUE);

        if (GR_atom(prodp) && GR_atom(prodq))
            {
                if (atomic_type(prodp) == atomic_type(prodq))
                    return(comparison(atomic_type(prodp),
                                       value_ptr(p),
                                       value_ptr(q))
                                       != Equal;
                }
            }
        else
            return(TRUE);
    }
```

```c
for (i = leftmost_son(prodp); i <= rightmost_son(prodp); i++)
    {
        test = different(son(p, i), son(q, i));
        if (test != FALSE)
            return(test);
    }
```
APPENDIX D - Auxiliary Functions

```c
); return(FALSE);
}

init_sys3(){
ME me_user;
/* extern PROCEDURE call_graphic_editor(); */
/* extern PROCEDURE save_graphic_attributes(); */
FILE *tool_file, *fopen();
char Buffer[100 + 1];
char home_dir[100];
/* struct passwd *user_pass; */

void
edit_graph(),
sort_psdi_components(),
clean_gedatransfile(),
enforce_consistency_on(),
enforce_consistency_off();

int
create_sh_query(),
print_psdi(),
/*
split_psdi();
*/
save_psdi(),
save_psdi_exit();

extern HeadPtr
prototype;

extern int
unique_id_count;

extern PRODUCTION
prod_prot,
prod_psdi_pair,
prod_op,
prod_data,
prod_no_component,
prod_op_spec,
prod_op_impl,
prod_type_impl,
prod_t_op_impl,
prod_op_impl_list_pair,
prod_input_list,
prod_inputs,
prod_output_list,
prod_outputs,
prod_state_list,
create_sh_query,
prod_type_decl,
prod_graph_null,
prod_graph,
prod_vertex_list_pair,
prod_optional_type_id,
prod_operator_id_pairs,
prod_alone_id_pair,
prod_a_decl,
prod_decl,
prod_stream,
prod_cc,
prod_constraints,
prod_a_constraint;

#ifdef SDE_DEBUG
/* debugging */
printf('Entering init_sys3\n');
#endif

/* create new menu */
me_user = me_open("CAPS-Cmds");
/* create five menu items */
(void) co_open(
  "save-psdi",
  save_psdi,
  co_valid
);

(void) co_open(
  "save-psdi-exit",
  save_psdi_exit,
  co_valid
);

#endif D_EDITOR
(void) co_open(
  "enforce-consistency_on",
  enforce_consistency_on,
  co_valid
);

(void) co_open(
  "enforce-consistency_off",
  enforce_consistency_off,
  co_valid
);
```

196
APPENDIX D - Auxiliary Functions

```c
print_psd1,
co_valid
);

/*
(void) co_open{
  "split-psd1",
  split_psd1,
  co_valid
);
*/

if (me_user != HR_NULL)
  { MenuItem("edit-graph", me_user);
    MenuItem("create-ab-query", me_user);
    MenuItem("print-psd1", me_user);
    MenuItem("split-psd1", me_user);
    MenuItem("save-psd", me_user);
    MenuItem("save-psd-exit", me_user);
  }
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
```

```c
/*
user_pass = getpwd();
strncpy(home_dir, user_pass->pwd_dir);
*/
strncpy(home_dir, getenv("CAPS_HOME"));
strcat(home_dir, "/bin/tool_location.txt");
clean_gerd.datatransfile();
if(tool_file = fopen(home_dir, "rt")) == NULL)
    printf("Tool location file not found.
   
   ");
else
    fscanf(tool_file, "ts", Buffer);
    while(strcmp(Buffer, "graph_viewer") != 0)   
        fscanf(tool_file, "ts", Buffer);
    fgets(Buffer, 100, tool_file);
    system(Buffer);
fclose(tool_file);
/* initialize the global variables */
prod_prot = op_search("Prot");
prod_psd1_mil = op_search("FsdM11");
prod_psd1_pair = op_search("FsdPair");
prod_op = op_search("Op");
prod_data = op_search("Data");
prod_no_component = op_search("NoComponent");
prod_op_spec = op_search("OperatorSpec");
prod_op_impl = op_search("OperatorImpl");
prod_type_impl = op_search("TypeImpl");
prod_t_op_spec = op_search("TopSpec");
prod_t_op_impl = op_search("TopImpl");
prod_op_impl_list_pair = op_search("OpImplListPair");
prod_input_list = op_search("InputsListPair");
prod_inputs = op_search("Inputs");
prod_output_list = op_search("OutputsListPair");
prod_outputs = op_search("Outputs");
prod_state_list_name = op_search("StateListName");
prod_state_list = op_search("StateListPath");
prod_type_decl = op_search("TypeDecl");
prod_graph_null = op_search("GraphNull");
prod_graph = op_search("Graph");
prod_vertex_list_pair = op_search("VertexListPair");
prod_optional_type_id = op_search("OptionalTypeId");
prod_operator_id_pair = op_search("OperatorIdPair");
prod_alone_id_pair = op_search("AlonePair");
prod_a_decl = op_search("ADecl");
prod_decl = op_search("Declarations");
prod_stream = op_search("Streams");
prod_cc = op_search("Cc");
prod_constraints = op_search("ConstraintsPair");
prod_a_constraint = op_search("ACOnstraint");
prototype = NULL;
Global_Type_List = NULL;
*/
/*
Global_Operators = NULLVALUE;
*/
/*
Global_Protot = NULLVALUE;
Global_Roots = NULLVALUE;
Global_Type_Decl = NULLVALUE;
Global_Undef_Ops = NULLVALUE;
Global_Undef_Type_Ops = NULLVALUE;
Global_Current_OP_Name = "";
unique_id_count = 1;
Global_Refresh_Graph_Viewer = true;
*/
default of editor is to have Enforce_Consistency on
*/
Global_Enforce_Consistency = true;
Global_Set_SideView = false;
printf("init.sys": Global_Set_SideView = false
   ");
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
```

```c
char * get_prototype_name()
{
  /*This procedure extracts the prototype name from the title of the
current buffer.*/
  char
  *name,
  *component_name,
```
APPENDIX D - Auxiliary Functions

```c
/* attempt to extract a psdl identifier from file name */
path_name = strdup(bu_file_name(br_buf(cur_browser)));
#else
printf("path_name = %s\n", path_name);
#endif

component_name = strrchr(path_name, '/');
if (component_name != NULL)
    component_name++; /* skip the '/' character */
else
    component_name = path_name;
#endif

/* debuging */
printf("get.prototype_name: component_name = %s\n", component_name);
#endif

if (strcmp(component_name, "") != 0)
    return(component_name);
else
    return("_default_proto_name");

init_sys]()
/
*/
This procedure assumes that the "prototype" data structure is not NULL. If
such is not the case, there is a discrepancy between the attributes file and
the input text file... execution is thus terminated.
*/

extern int Process_File();
char
*component_name,
*get.prototype_name();
```

```c
ATREE
  temp_tree,
  atree = bu_atree(br_buf(cur_browser));
#endif

PROD_INSTANCE
Get_Id(),
Sort_psd1_Components(),
Make_Proto(),
temp_p;
#endif

PROD_INSTANCE
Change_view();
#endif

void
Add_Root_Op();
#endif

boolean
keep_searching;
#endif

/* debuging */
printf("Entering init_sys6\n");
#endif

if( Process_File() != TRUE)
{
    printf("INPUT FILES ARE CORRUPTED. PLEASE CHECK\n");
    exit(777);
}
else
    printf("INPUT FILES SUCCESSFULLY LOADED\n");
#endif

Change_view(SSLstring("SDE_VIEW"), SSL_true);
Global_Set_SdevView = true;
printf("init_sys6: Global_Set_SdevView = true\n");

component_name = strdup(get.prototype_name());

/* add operator template with name == component_name */
if (production(psd1_components_from_Prot(Global_Prot0)? prod_psd1_pair &&
            strcmp(component_name, "_default_proto_name") != 0))
    {
    #ifdef SDE_DEBUG3
    /* debuging */
    printf("init_sys6: psdl_components_from_Prot(Global_Prot0) == PsdlNil\n");
#endif
    Add_Root_Op(component_name);
    temp_p = graph_from_operatorImpl{
        operator_impl_from_Op(
            component_from_Psd1Pair(
                psdl_components_from_Prot(Global_Prot0)));
        move_selection(atree, one_point_selection(temp_p));
        br_set_insert_pt_to_selection(cur_browser);
        cmd_cond_modifies(cur_browser, cur_buffer);
        br_paint_all();
    }
    else
```
APPENDIX D - Auxiliary Functions

/*
 * if ((production[son(son(Global_Proto, 1), 1)]) == prod_no_component) &&
 */
if ((production[component_from_Psd1Pair(
    psdl_components_from_Prot(Global_Proto)]) == prod_no_component) &&
    (strcmp(component_name, "_default_proto_name") != 0))
{
    ifdef SDE_DEBUG
    /* debugging*/
    printf("init_sys6: component_from_Psd1Pair(psdl_components_from_Prot(Global_Proto)) == NoComponent\n");
    endif

    Add_Root_Op(component_name);
    temp_p = graph_from_OperatorImpl(
        operator_impl_from_Op(
            component_from_Psd1Pair(
                psdl_components_from_Prot(Global_Proto))));
    move_selection(atree, one_point_selection(temp_p));
    br_set_insert_pt_to_selection(cur_browser);
    cmd_cond_modifies(cur_browser, cur_buffer);
    br_paint_all();
}
else
{
    /* sort the components alphabetically, putting all Types before Ops */
    move_selection(atree, one_point_selection(Global_Proto));
    /*
    * temp_p = son(Global_Proto, 1);
    */
    temp_p = psdl_components_from_Prot(Global_Proto);
    temp_p = Sort_Psd1_Components(temp_p);
    temp_tree = tree_to_atree(Make_Proto(temp_p));
    if (context(temp_tree) != context(atree))
        insert_coercion(atree, temp_tree);
    else
        swap_selections(temp_tree, atree);
    establish_resting_place(atree);
    rm_atree(temp_tree);
    br_set_insert_pt_to_selection(cur_browser);
    cmd_cond_modifies(cur_browser, cur_buffer);
    br_paint_all();
    if (strcmp(component_name, "_default_proto_name") != 0) {
    /* position cursor at component with name == component_name */
    /*
    * temp_p = son(Global_Proto, 1);
    */
    temp_p = psdl_components_from_Prot(Global_Proto);
    keep_searching = true;
    while ((production(temp_p) == prod_psd1_pair) && keep_searching)
        if (strcmp(component_name,
            /*
            */
            StrValue(son(son(temp_p, 1), 1)),
        1))
        {
            str0_to_str_ro(
                StrValue(
                    Get_Id(
                        id_from_Op(
                            component_from_Psd1Pair(temp_p))))
                    )
        }
        
        if (keep_searching) {
    temp_p = component_from_Psd1Pair(temp_p);
    if (production(temp_p) == prod_op) {
        if (IntValue(Get_Impl_Form(
            operator_impl_from_Op(temp_p))) > 0)
            temp_p = graph_from_OperatorImpl(
                operator_impl_from_Op(temp_p));
        move_selection(atree, one_point_selection(temp_p));
        br_set_insert_pt_to_selection(cur_browser);
        cmd_cond_modifies(cur_browser, cur_buffer);
        br_paint_all();
    }
    else
    {
        temp_p = component_from_Psd1Pair(
            psdl_components_from_Prot(Global_Proto));
        move_selection(atree, one_point_selection(temp_p));
        br_set_insert_pt_to_selection(cur_browser);
        cmd_cond_modifies(cur_browser, cur_buffer);
        br_paint_all();
    }
    }
    
    /* call Refresh_Graph_Viewer to display the graph */
    /* (MTS) 1/12/95 */
    Refresh_Graph_Viewer();

    }

    }

    
    
    /* change the view from DEFAULT to SDE_VIEW */
APPENDIX D - Auxiliary Functions

if (BoolValue(Empty_Graph(temp_p)))
{
    /* need to replace empty operator impl by OpImplNull
       and reset cursor position to current point */

    Free_Linked_List(current_pos_trace);
    current_pos_trace = NULL;
}

while (!at_top(temp_p))
{
    temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
    temp_head->item_number = son_number(temp_p);
    temp_head->next = current_pos_trace;
    current_pos_trace = temp_head;

    temp_p = father(temp_p);
}

Remove_Empty_Graph();

/* save graphic information */
save_graphic_file();

/* get the prototype name */
component_name = strdup(get_prototype_name());

/* create the output file name */
sprintf(buffer, "%s*.psdl", component_name, ".psdl");

if (production(pSDL_components_from_Prot(Global_proto)) == prod_pSDL_nil)
{
    write_error_string("empty prototype, nothing to be saved");
}
else
{
    if (IntValue(save_as)
        SSLstring("Text"),
        SSLstring(buffer),
        SSLstring("BASEVIEW")
        SSLstring("OUTPUTVIEW")
    )
    {
        ++1;
        sprintf(buffer, "%s*.psdl", "can't write file " , component_name, ".psdl");
        write_error_string(buffer);
        result = 1;
    }
    if (split_pSDL() == 1)
    {
        result = 1;
    }
}

/* move cursor back to original position */
temp_p = Global_proto;

temp_head = current_pos_trace;
while (temp_head != NULL)
APPENDIX D - Auxiliary Functions

```c
{ temp_p = son(temp_p, (temp_head->item_number));
temp_head = temp_head->next;
}
p = temp_p;
Free_Linked_List(current_pos_trace);

temp_p = one_point_selection(p);
move_selection(atree, temp_p);
br_set_insert_pt_to_selection(cur_browser);
cmd_cond_modifies(cur_browser, cur_buffer);
br_paint_all();

return(result);
}

int print_psd1()
{
 PROD_INSTANCE
 Save_as()
 result;

char buffer[100];
char *
component_name, 
*get_prototype_name();

/* get the prototype name */
component_name = strdup(get_prototype_name());
sprintf(buffer, "%s", component_name, ".psdl.printfile");

result = Save_as(
 SSString("Text"),
 SSString(buffer),
 SSString("SAVVIEW")
 */
 SSString("OUTPUT_VIEW"
 */;

if (IntValue(result) != 0)
{
 write_error_string("can't save psdl, print failed");
br_paint_all();
return(1);
}

sprintf(buffer, "%s\%s\%s", "lpr ", component_name, ".psdl.printfile");
if (IntValue(result) != 0)
{
 write_error_string("can't save psdl, print failed");
br_paint_all();
}
return(1);
}

system(buffer);
return(IntValue(result));

/*-----------------------------------------------*/
int create_sb_query()
{
 PROD_INSTANCE
 Save_selection_to_file()
 temp_p;

ATREE
 atree = bu_atree(br_buf(cur_browser));
char buffer[100];
intprint_psd1_spec(),
result;

LINKED_LIST
 temp_head, current_pos_trace = NULL;
void
Free_Linked_List();

char *
component_name, 
*get_prototype_name();

/* remember the cursor position */
p = selection_sec(SE_selection(bu_atree(br_buf(cur_browser))));

temp_p = p;
current_pos_trace = NULL;

/* Note: need to delete (production(temp_p)!=prod_data) in
the following boolean expression if type specs are
not allowed in sb query Shing 8/8/94 */
while (at_top(temp_p) && (production(temp_p)!=prod_op)
 && (production(temp_p)!=prod_data))
{
 temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
temp_head->item_number = son_number(temp_p);
temp_head->next = current_pos_trace;
current_pos_trace = temp_head;
temp_p = father(temp_p);
}
if (at_top(temp_p))
{
 write_error_string("must select component before invoking create sb-query");
Free_Linked_List(current_pos_trace);
return(1);
}
```
APPENDIX D - Auxiliary Functions

```c
/* move selection and cursor to select the whole component */
move_selection(atree, one_point_selection(temp_p));
br_set_insert_pt_to_selection(cu_buffer);
cmd_cond_modifies(cu_browser, cu_buffer);
br_paint_all();

component_name = strdup(get_prototype_name());

sprintf(buffer, "%s", component_name, "pdsl ámbex’");
result = print_paddl_spec(buffer);

/* move cursor back to original position */
temp_head = current_pos_trace;
while (temp_head != NULL)
{
    temp_p = son(temp_p, (temp_head->item_number));
    temp_head = temp_head->next;
}
p = temp_p;
Free_Linked_List(current_pos_trace);

temp_p = one_point_selection(p);
move_selection(atree, temp_p);
br_set_insert_pt_to_selection(cu_browser);
cmd_cond_modifies(cu_browser, cu_buffer);
br_paint_all();

return(result);
}

int split_padt()
{
    PROD_INSTANCE
    Empty_Graph(),
    p,
    temp_p;

    ATREE
    atree = bu_atree(br_buf(cu_browser));

    temp_p = pdsl_components_from_prot(Global_Proto);
    temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
    temp_head->item_number = son_number(temp_p);
    temp_head->next = current_pos_trace;
    current_pos_trace = temp_head;
    temp_p = father(temp_p);

    while (!at_top(temp_p))
    {
        temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
        temp_head->item_number = son_number(temp_p);
        temp_head->next = current_pos_trace;
        current_pos_trace = temp_head;
    }
}

void Free_Linked_List()
{
    int result = 0;
    
    /* component_name,
       *get_prototype_name();
       
    component_name = strdup(get_prototype_name());
       */

    /* remember the cursor position */
    p = selection Apex(SE_selection(br_atree(br_buf(cu_browser))));
    temp_p = p;
    current_pos_trace = NULL;

    while (!at_top(temp_p) && production(temp_p) != prod_op_impl)
    {
        temp_head = (LINKED LIST) malloc(sizeof(LINKED_LIST NODE));
        temp_head->item_number = son_number(temp_p);
        temp_head->next = current_pos_trace;
        current_pos_trace = temp_head;
        
        temp_p = father(temp_p);
    }

    if (production(temp_p) == prod_op_impl)
    {
        if (BoolValue(Empty Graph(temp_p)))
        {
            /* need to replace empty operator impl by OptImplNull */
            Free_Linked_List(current_pos_trace);
            current_pos_trace = NULL;
        }
        
        while (!at_top(temp_p))
        {
            temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
            temp_head->item_number = son_number(temp_p);
            temp_head->next = current_pos_trace;
            current_pos_trace = temp_head;
        }
    }
}

while (production(temp_p) == prod_padt_pair)
{
    p = component_from_paddl_pair(temp_p);
    if ((production(p) == prod_op) || (production(p) == prod_data))
    {
        /* move selection and cursor to select the whole component */
        move selection(atree, one_point_selection(p));
        br_set_insert_pt_to_selection(cu_browser);
        cmd_cond_modifies(cu_browser, cu_buffer);
        br_paint_all();
    }
    else
    {
        /* component_name,
           *get_prototype_name();
           
        component_name = strdup(get_prototype_name());
           */

        /* remember the cursor position */
        p = selection Apex(SE_selection(br_atree(br_buf(cu_browser))));
        temp_p = p;
        current_pos_trace = NULL;

        while (!at_top(temp_p) && production(temp_p) != prod_op_impl)
        {
            temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
            temp_head->item_number = son_number(temp_p);
            temp_head->next = current_pos_trace;
            current_pos_trace = temp_head;
            
            temp_p = father(temp_p);
        }

        if (production(temp_p) == prod_op_impl)
        {
            if (BoolValue(Empty Graph(temp_p)))
            {
                /* need to replace empty operator impl by OptImplNull */
                Free_Linked_List(current_pos_trace);
                current_pos_trace = NULL;
            }
            
            while (!at_top(temp_p))
            {
                temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
                temp_head->item_number = son_number(temp_p);
                temp_head->next = current_pos_trace;
                current_pos_trace = temp_head;
            }
        }
    }
}

int split_padt()
{
    PROD_INSTANCE
    Empty_Graph(),
    p,
    temp_p;

    ATREE
    atree = bu_atree(br_buf(cu_browser));

    temp_p = pdsl_components_from_prot(Global_Proto);
    temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
    temp_head->item_number = son_number(temp_p);
    temp_head->next = current_pos_trace;
    current_pos_trace = temp_head;
    temp_p = father(temp_p);

    while (!at_top(temp_p))
    {
        temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
        temp_head->item_number = son_number(temp_p);
        temp_head->next = current_pos_trace;
        current_pos_trace = temp_head;
    }
}

void Free_Linked_List()
{
    int result = 0;
    
    /* component_name,
       *get_prototype_name();
       
    component_name = strdup(get_prototype_name());
       */

    /* remember the cursor position */
    p = selection Apex(SE_selection(br_atree(br_buf(cu_browser))));
    temp_p = p;
    current_pos_trace = NULL;

    while (!at_top(temp_p) && production(temp_p) != prod_op_impl)
    {
        temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
        temp_head->item_number = son_number(temp_p);
        temp_head->next = current_pos_trace;
        current_pos_trace = temp_head;
        
        temp_p = father(temp_p);
    }

    if (production(temp_p) == prod_op_impl)
    {
        if (BoolValue(Empty Graph(temp_p)))
        {
            /* need to replace empty operator impl by OptImplNull */
            Free_Linked_List(current_pos_trace);
            current_pos_trace = NULL;
        }
        
        while (!at_top(temp_p))
        {
            temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
            temp_head->item_number = son_number(temp_p);
            temp_head->next = current_pos_trace;
            current_pos_trace = temp_head;
        }
    }
}
```

202
APPENDIX D - Auxiliary Functions

```c
int print_psd1_spec(fn)
    char* fn;
{
    result = Print_Instance(result, 
        Save_selection_to_file( 
            SLLstring("Text"),  
            SLLstring(fn), 
            SLLstring("SPECONLYVIEW") 
        ));
    return(IntValue(result));
}
```

```c
/*---------------------------------*/
int print_psd1_impl(fn)
    char* fn;
{
    result = Print_Instance(result, 
        Save_selection_to_file( 
            SLLstring("Text"),  
            SLLstring(fn), 
            SLLstring("IMPLEONLYVIEW") 
        ));
    return(IntValue(result));
}
```

```c
/*---------------------------------*/
int save_psd1_exit()
{
    if (save_psd1() == 0)
        finish(strdup(""));
    return(0);
}
```

```c
/*---------------------------------*/
void enforce_consistency_on()
{
    extern boolean Global_Enforce_Consistency;
    ifndef SDE_DEBUG1
        printf("Entering enforce_consistency_on
\n");
    endif
    Global_Enforce_Consistency = true;
    ifndef SDE_DEBUG1
        printf("Leaving enforce_consistency_on
\n");
    endif
}
```

```c
/*---------------------------------*/
void enforce_consistency_off()
{
    extern boolean Global_Enforce_Consistency;
    ifndef SDE_DEBUG1
        printf("Entering enforce_consistency_off
\n");
    endif
    Global_Enforce_Consistency = false;
}
```
APPENDIX D - Auxiliary Functions

```c
#define SDE_DEBUG

static BUFFER
BUF;
BUF;

if (at_top(temp)) temp = selection_apex(SE_selection(atree));
remove_extra_placeholders(atree, &temp);
s = one_point_selection(temp);
move_selection(atree, s);
br_set_insert_pt_to_selection(cur_browser);
cond_modifies(cur_browser, cur_buffer);
br_paint_all();

PROCEDURE prepareBufferToClear(buf)
BUFFER buf;

extern BROWSER
br_list_head;
BROWSER
b;

/* debuging */
printf("Entering prepareBufferToClear

endif

/* test buffer for completing term */
if (is_completing_term(bu_atree(bu) = root)) return;

/* Put completing production in bu and rm_atree that was there */
top(bu_atree(bu));

*atree = completing_atree(bu_node(bu));
prepareBufferToClear(bu);
prepareToSwap(bu_atree(bu));
buf_atree(bu, atree);
restore_tree_after_selection_modification(atree);
/*
*atree = mk_atree(tree_to_abody(test_subtree()));
/*
old_atree = bu_atree(bu);
atree = mk_atree(tree_to_abody_reuse(old_atree = root));
prepareToSwap(old_atree);
buf_atree(bu, atree);
restore_tree_after_selection_modification(atree);
/*
*temp = selection_apex(SE_selection(atree));
**
*temp = one_step_forward(temp);
*/
for (i=1:i<3;i++)
{"temp = one_step_forward_without_descent(temp);
*while (is_a_list_nil(temp) || is_an_opt_empty(temp) ||
  InTextBuffer(at_texts_buffer[cur_table][atree], cur_view, temp)
  && !at_top(temp)
  );
*/
```

void Link_Stream(p, h)
ST_PTR p;
HeadPtr h;

/* debuging */
printf("Entering Link_Stream

endif

p->next = h->stream_list;
h->stream_list = p;
*/
```

void Link_Operator(p, h)
OPNodePtr p;
HeadPtr h;

/* This procedure takes as input two pointers: "p" and "h". "p" points to a
vertex head node which is about to be inserted in the vertex list pointed

* /
```
APPENDIX D - Auxiliary Functions

```c
if (production(p) == op)
    
    h = Mark_Operator(key);
#else
    if (production(p) == psdl_pair)
        while (production(p) != psdl_nil)
        
        printf("list of components detected\n");
#endif
#endif /* CANCEL_TEMP_DELETE */

/* get access to the 'deleted' buffer's abstract tree */
the_tree = bu_atree{ deleted };
#endif /* CANCEL_TEMP_DELETE */
```

```c
if (production(p) == op)
    
    key = (int)p;
#endif
#endif /* CANCEL_TEMP_DELETE */

/* get access to the top_node*/
p = the_tree->root;
#endif
```

```c
if (p == NULLVALUE) return;
```
APPENDIX D - Auxiliary Functions

```c

/* global_root_store_init()
 */
#endif worldwide_DEBUG_1

/* global_root_store_fetch()
 */
#endif worldwide_DEBUG_1

/* global_root_store_delete(p, a, v)
 */
#endif worldwide_DEBUG_1

/* global_operators_store_init()
 */
#endif worldwide_DEBUG_1
```

206
APPENDIX D - Auxiliary Functions

```c
#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_operators_store_delete\n");
    printf("Leaving global_operators_store_delete\n");
#endif

/*-----------------------------------------------*/
global_operators_store_delete(p, a)
    PROD_INSTANCE p;
    ATTR_NO a;
{
#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_operators_store_delete\n");
    printf("Leaving global_operators_store_delete\n");
#endif
}

/*-----------------------------------------------*/
global_operators_store_insert(p, a, v)
    PROD_INSTANCE p;
    ATTR_NO a;
    PROD_INSTANCE v;
{
#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_operators_store_insert\n");
#endif
    Global_Operators = v;  /* Global_Operators is global variable */
#endif SDE_DEBUG1
    /* debugging */
    printf("leaving global_operators_store_insert\n");
#endif
}

/*-----------------------------------------------*/
global_streams_store_init()
#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_streams_store_init\n");
    printf("Leaving global_streams_store_init\n");
#endif
}

/*-----------------------------------------------*/
PROD_INSTANCE global_streams_store_fetch()
#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_streams_store_fetch\n");
    printf("Leaving global_streams_store_fetch\n");
#endif
}

/*-----------------------------------------------*/
global_streams_store_delete(p, a)
    PROD_INSTANCE p;
    ATTR_NO a;
{
#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_streams_store_delete\n");
#endif
}

/*-----------------------------------------------*/
global_streams_store_insert(p, a, v)
    PROD_INSTANCE p;
    ATTR_NO a;
    PROD_INSTANCE v;
{
#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_streams_store_insert\n");
#endif
}
```
APPENDIX D - Auxiliary Functions

```c
GlobalStreams = v; /* GlobalStreams is global variable */

#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_streams_store_insert\n\n");
#endif

/*--------------------------------------------*/
global_states_store_init()

#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_states_store_init\n\n");
#endif

/*--------------------------------------------*/
PROD_INSTANCE global_states_store_fetch()

#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_states_store_fetch\n\n");
#endif

/*--------------------------------------------*/
global_states_store_delete(p, a)
    PROD_INSTANCE p;
    ATTR_NO a;

#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_states_store_delete\n\n");
#endif

/*--------------------------------------------*/
global_states_store_insert(p, a, v)
    PROD_INSTANCE p;
    ATTR_NO a;
    PROD_INSTANCE v;

#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_states_store_insert\n\n");
#endif

/*--------------------------------------------*/
GlobalStates = v; /* GlobalStates is global variable */

#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_states_store_init\n\n");
#endif

/*--------------------------------------------*/
PROD_INSTANCE global_edges_store_init()

#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_edges_store_init\n\n");
#endif

/*--------------------------------------------*/
PROD_INSTANCE global_edges_store_fetch()

#ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering global_edges_store_fetch\n\n");
#endif
```

208
APPENDIX D - Auxiliary Functions

/------------------------------------------*/
global_edges_store_delete(p, a)
  PROD_INSTANCE p;
  ATTR_NO a;
{
  *debugging */
  printf("Entering global_edges_store_delete\n");
  printf("Leaving global_edges_store_delete\n");
  *endif

} /* global_edges_store_insert(p, a, v) */
  PROD_INSTANCE p;
  ATTR_NO a;
  PROD_INSTANCE v;
{
  *debugging */
  printf("Entering global_edges_store_insert\n");
  *endif

  Global_Edges = v; /* Global_Edges is global variable */

  *debugging */
  printf("Entering global_edges_store_insert\n");
  printf("Leaving global_edges_store_insert\n");
  *endif

  /* CANCEL_TEMP_DELETE */
/------------------------------------------*/

*debugging */
  printf("Entering global_undef_ops_store_init\n");
  printf("Leaving global_undef_ops_store_init\n");
*endif

/------------------------------------------*/

*debugging */
  printf("Entering global_undef_ops_store_fetch\n");
  printf("Leaving global_undef_ops_store_fetch\n");
*endif

/------------------------------------------*/

*debugging */
  printf("Entering global_undef_ops_store_delete\n");
  printf("Leaving global_undef_ops_store_delete\n");
*endif

/------------------------------------------*/

global_undef_ops_store_insert(p, a, v)
  PROD_INSTANCE p;
  ATTR_NO a;
  PROD_INSTANCE v;
{
  *debugging */
  printf("Entering global_undef_ops_store_insert\n");
  *endif

  Global_Undef_Ops = v; /* Global_Undef_Ops is global variable */

  *debugging */
  printf("Entering global_undef_ops_store_insert\n");
  printf("Leaving global_undef_ops_store_insert\n");
  *endif

  if (v == NULLVALUE) return;

  temp_new_p = v;
  while(!BoolValue(IsNull(temp_new_p)))
  {
    printf("global_undef_ops_store_insert: id = %s\n", 
      str0_to_str_ro(StrValue(Get_Id(FirstElement[temp_new_p]))));
    temp_new_p = IdSetTail[temp_new_p];
  }
  *debugging */
  printf("Leaving global_undef_ops_store_insert\n");
  *endif

/------------------------------------------*/

*debugging */
  printf("Entering global_undef_type_ops_store_init\n");
  printf("Leaving global_undef_type_ops_store_init\n");
*endif

/------------------------------------------*/

*debugging */
  printf("Entering global_undef_type_ops_store_fetch\n");
  printf("Leaving global_undef_type_ops_store_fetch\n");
*endif

/------------------------------------------*/

*debugging */
  printf("Entering global_undef_type_ops_store_delete\n");
  printf("Leaving global_undef_type_ops_store_delete\n");
*endif

/------------------------------------------*/

*debugging */
  printf("Entering global_undef_type_ops_store_delete\n");
  printf("Leaving global_undef_type_ops_store_delete\n");
*endif

/------------------------------------------*/

*/

*/

*/

*/

*/

*/

*/

*/

*/

*/

*/

*/

*/

*/

*/

*/

*/

209
APPENDIX D - Auxiliary Functions

```c
#ifdef SDE_DEBUG_1
    /* debugging */
    printf("Entering global_undef_type_ops_store_insert\n");
#endif

#define SDE_DEBUG_2
    /* Global.Undef.Type.Ops is global variable */
#endif

#ifdef SDE_DEBUG_1
    /* debugging */
    printf("leaving global_undef_type_ops_store_insert\n");
#endif

#define SDE_DEBUG_2
    /* debugging */
    printf("leaving operator_id_store_init\n");
#endif

#define SDE_DEBUG_2
    /* debugging */
    printf("leaving operator_id_store_fetch\n");
#endif

operator_id_store_delete(p, a)

operator_id_store_insert(p, a, v)
```

```c
if (null)
    /* not used here */
#endif

if (null)
    "string_to_string(StrValue(Get_Id[IId_from_Op(p)]))"
    printf("OPERATOR_STORE_DELETE: OPERATOR ->\" NOT FOUND IN STRUCTURE\n", name);
    /* the following statement is commented out, may cause memory leak */
    /* free(name); */
#endif

if (null)
    printf("Leaving operator_id_store_delete\n");
#endif

operator_id_store_fetch(p, a)
operator_id_store_insert(p, a, v)
```
APPENDIX D - Auxiliary Functions

```c
#ifdef SDE_DEBUG_1
    /* debugging */
    printf("Entering operator_id_store_insert\n\n");
#endif

    /* the following command is commented out for ease of debugging 1/21/94 */
    /* Output_Structure(); */

    /*
     * mark_killed_operators();
     */

    if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
    /* debugging */
    printf("Operator_id_store_insert: Op Name = %s\n", str0_to_str_ro(StrValue(Get_Id(id_from_op(p)))));
#endif

    h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);
    prod_no = (int)p;
    new_name = str0_to_str_ro(StrValue(Get_Id(v)));

#ifdef SDE_DEBUG_2
    /* debugging */
    printf("INSERT Operator %s %d\n", new_name, prod_no);
#endif

    if (h == NULL)
    {
        /* need to make new header node */
        h = Make_Operator_Header(new_name, NULL, NULL, Get_Unde_link();
        Head_To_Structure(h);
    }
    else
    {
        /* operator already in the list, update name and prod_no */
        if (strcmp(h->name, "") != 0)
        {
            free(h->name);
            free(h->ada_op_name);
            h->name = strdup(new_name);
            h->ada_op_name = strdup(new_name);
            h->prod_no = h->prod_no;
        }
    }

#ifdef SDE_DEBUG_1
    /* debugging */
    printf("Leaving operator_id_store_insert\n\n");
#endif
#endif

    states_id_store_init()
    {
        #ifdef SDE_DEBUG_1
            /* debugging */
            printf("Entering states_id_store_init\n\n");
            printf("Leaving states_id_store_init\n\n");
        #endif
    }

    PROD_INSTANCE states_id_store_fetch()
    {
        #ifdef SDE_DEBUG_1
            /* debugging */
            printf("Entering states_id_store_fetch\n\n");
            printf("Leaving states_id_store_fetch\n\n");
        #endif
    }

    states_id_store_delete(p, a)
    {
        PROD_INSTANCE p;
        ATTR_NO a;

        #ifdef SDE_DEBUG_1
            /* debugging */
            printf("Entering states_id_store_delete\n\n");
            printf("Leaving states_id_store_delete\n\n");
        #endif
    }

    states_id_store_insert(p, a, v)
    {
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;

        HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

        void
        Move_To_Structure_Front();

        #ifdef SDE_DEBUG_1
            /* debugging */
            printf("Entering states_id_store_insert\n\n");
        #endif
        if (v == NULLVALUE) return;

        #ifdef SDE_DEBUG_1
            /* debugging */
            printf("States_id_store_insert: Op Name = %s\n", str0_to_str_ro(StrValue(Get_Id(id_from_op(p)))));
        #endif

        h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

        if (h != NULL)
        {
            /*
             * enforce most-recently-used-first rule */
            if (prototype != h)
            {
                /* move header node to front of list */
                Move_To_Structure_Front(h);
            }
            #ifdef SDE_DEBUG_1
                printf("Leaving operator_id_store_insert\n\n");
            #endif
        }
    }
```

211
APPENDIX D - Auxiliary Functions

h->state_id_set = v;

/* enforce most-recently-used-first rule */
if (prototype != h)
{
  /* move header node to front of list */
  Move_To_Structure_Front(h);
}
else
{
  printf("states_ids_store_insert: operator not found\n");
}
#endif

#define GDK_DEBUG_1
printf("Leaving states_ids_store_insert\n");
#endif

/**----------------------------------------*/
inh_input_ids_store_init()
{
#define GDK_DEBUG_1
  /* debugging */
  printf("Entering inh_input_ids_store_init\n");
  printf("Leaving inh_input_ids_store_init\n");
#endif

/**----------------------------------------*/
PROD_INSTANCE inh_input_ids_store_fetch()
{
#define GDK_DEBUG_1
  /* debugging */
  printf("Entering inh_input_ids_store_fetch\n");
  printf("Leaving inh_input_ids_store_fetch\n");
#endif

/**----------------------------------------*/
inh_input_ids_store_delete(p, a)

#define GDK_DEBUG_1
  /* debugging */
  printf("Entering inh_input_ids_store_delete\n");
  printf("Leaving inh_input_ids_store_delete\n");
#endif

/**----------------------------------------*/
inh_input_ids_store_insert(p, a, v)

#define GDK_DEBUG_1
  /* debugging */
  printf("Entering inh_input_ids_store_insert\n");
  printf("Leaving inh_input_ids_store_insert\n");
#endif

HeadPtr h, Find_Header_Node_from_Op_or_TopImpl_or_TopSpec();
void Move_To_Structure_Front();
APPENDIX D - Auxiliary Functions

```c
#define SDE_DEBUG
/* debugging */
printf("Entering inh_output_ids_store_delete\n");
printf("Leaving inh_output_ids_store_delete\n");
#endif

/*----------------------------------------*/
inh_output_ids_store_insert(p, a, v)
  PROD_INSTANCE p;
  ATTR_NO a;
  PROD_INSTANCE v;
{
  HeadPtr h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_First();

#define SDE_DEBUG
/* debugging */
printf("Entering inh_output_ids_store_insert\n");
#endif

if (v == NULLVALUE) return;

#define SDE_DEBUG
/* debugging */
printf("Inh_output_ids_store_insert: Op Name = %s\n",
  str2to_str_ro(StrValue(Get_Id(id_from_op(p)))));
#endif

h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);
if (h != NULL)
{
  h->inh_output_id_set = v;
  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    move header node to front of list */
    Move_To_Structure_First(h);
  }
  else
  {
    printf("inh_output_ids_store_insert: operator not found\n");
  }
#endif SDE_DEBUG

printf("Leaving inh_output_ids_store_insert\n");
#endif

input_error_store_delete(p, a)
  PROD_INSTANCE p;
  ATTR_NO a;
{
#define SDE_DEBUG
/* debugging */
printf("Entering input_error_store_delete\n");
printf("Leaving input_error_store_delete\n");
#endif

/*----------------------------------------*/
input_error_store_insert(p, a, v)
  PROD_INSTANCE p;
  ATTR_NO a;
  PROD_INSTANCE v;
{
  HeadPtr h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_First();

#define SDE_DEBUG
/* debugging */
printf("Entering input_error_store_insert\n");
#endif

if (v == NULLVALUE) return;

#define SDE_DEBUG
/* debugging */
printf("Input_error_store_insert: Op Name = %s\n",
  str2to_str_ro(StrValue(Get_Id(id_from_op(p)))));
printf("Input_error_store_insert: Input_error = %s\n",
  BoolValue(v) ? "true": "false");
#endif

h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);
if (h != NULL)
{
  h->input_error = BoolValue(v);
  /* enforce most-recently-used-first rule */
  if (prototype != h)
```
APPENDIX D - Auxiliary Functions

```c
/* move header_node to front of list */
Move_To_Structure_Front(h);
}
}
else
{
  printf("input_error_store_insert: operator not found\n");
}
#endif

/* output_error_store_init() */

#endif

/* output_error_store_fetch() */

#endif

*output_error_store_delete(p, a)
  PROD_INSTANCE p;
  ATTR_NO a;
{
  #ifdef SDE_DEBUG
    /* debugging */
    printf("Entering output_error_store_delete\n");
    printf("Leaving output_error_store_delete\n");
  #endif
}

output_error_store_insert(p, a, v)
  PROD_INSTANCE p;
  ATTR_NO a;
  PROD_INSTANCE v;
  FilePtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();
  void
  Move_To_Structure_Front();
{
  #ifdef SDE_DEBUG
    /* debugging */
    printf("Entering output_error_store_insert\n");
  #endif

  #ifdef SDE_DEBUG
    /* debugging */
    printf("output_error_store_insert: Op Name = %s\n");
    str_to_str_ro(ListValue(Get_IdIdx_from_Op(p)));
    printf("output_error_store_insert: Output Error = %s\n",
      BoolValue(v) ? "true" : "false");
  #endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);
  if (h != NULL)
    h->output_error = BoolValue(v);
  else
    printf("output_error_store_insert: operator not found\n");

  #ifdef SDE_DEBUG
    /* debugging */
    printf("Entering output_error_store_delete\n");
    printf("Leaving output_error_store_delete\n");
  #endif
}

*inh_input_decl_store_init() */

#endif

/* inh_input_decl_store_fetch() */

#endif

*inh_input_decl_store_delete(p, a)
  PROD_INSTANCE p;
  ATTR_NO a;
{
  #ifdef SDE_DEBUG
    /* debugging */
    printf("Entering inh_input_decl_store_delete\n");
    printf("Leaving inh_input_decl_store_delete\n");
  #endif
}
```

214
APPENDIX D - Auxiliary Functions

PROC_INSTANCE inh_output_decl_store_fetch()
{
  ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_output_decl_store_fetch\n");
  printf("Leaving inh_output_decl_store_fetch\n");
  ifndef

  INH_INPUT_DECL_STORE_INSERT(p, a, v)
  PROD_INSTANCE p;
  ATTR_NO a;
  PROD_INSTANCE v;

  
  HeadPtr
  h;
  Find_Header_Node_from_Op_or_TooImpl_or_TooSpec();

  void
  Move_To_Structure_Front();

  ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_input_decl_store_insert\n");
  ifndef

  if (v == NULLVALUE) return;

  ifdef SDE_DEBUG_1
  /* debugging */
  printf("Inh_input_decl_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p))));
  ifndef

  h = Find_Header_Node_from_Op_or_TooImpl_or_TooSpec(p);
  if (h != NULL)
  { h->inh_input_decl = v;
    /* enforce most-recently-used-first rule */
    if (prototype != h)
    { /* move header_node to front of list */
      Move_To_Structure_Front(h);
    }
  }
  else
  { printf("inh_input_decl_store_insert: operator not found\n");
  }

  ifdef SDE_DEBUG_1
  printf("Leaving inh_input_decl_store_insert\n");
  ifndef

  } /*-----------------------------------------------*/

  INH_OUTPUT_DECL_STORE_INIT()
  { ifdef SDE_DEBUG_1
    /* debugging */
    printf("Entering inh_output_decl_store_init\n");
    printf("Leaving inh_output_decl_store_init\n");
  ifndef

  } /*-----------------------------------------------*/

  INH_OUTPUT_DECL_STORE_DELETE(p, a)
  PROD_INSTANCE p;
  ATTR_NO a;

  ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_output_decl_store_delete\n");
  printf("Leaving inh_output_decl_store_delete\n");
  ifndef

  } /*-----------------------------------------------*/

  INH_INPUT_DECL_STORE_INSERT(p, a, v)
  PROD_INSTANCE p;
  ATTR_NO a;
  PROD_INSTANCE v;

  
  HeadPtr
  h;
  Find_Header_Node_from_Op_or_TooImpl_or_TooSpec();

  void
  Move_To_Structure_Front();

  ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_input_decl_store_insert\n");
  ifndef

  if (v == NULLVALUE) return;

  ifdef SDE_DEBUG_1
  /* debugging */
  printf("Inh_input_decl_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p))));
  ifndef

  h = Find_Header_Node_from_Op_or_TooImpl_or_TooSpec(p);
  if (h != NULL)
  { h->inh_output_decl = v;
    /* enforce most-recently-used-first rule */
    if (prototype != h)
    { /* move header_node to front of list */
      Move_To_Structure_Front(h);
    }
  }
  else
  { printf("inh_input_decl_store_insert: operator not found\n");
  }

  ifdef SDE_DEBUG_1
  printf("Leaving inh_input_decl_store_insert\n");
  ifndef

  } /*-----------------------------------------------*/
# APPENDIX D - Auxiliary Functions

```c
{
    printf("inh_output decl_store insert: operator not found\n"),
}
#endif
#endif

met_error_store_init()
{
    ifdef SDE_DEBUG_1
    /* debugging */
    printf("Entering met_error_store_init\n");
    printf("Leaving met_error_store_init\n");
    endif
}

/*------*/

met_error_store_fetch()
{
    ifdef SDE_DEBUG_1
    /* debugging */
    printf("Entering met_error_store_fetch\n");
    printf("Leaving met_error_store_fetch\n");
    endif
}

/*------*/

met_error_store_delete(p, a)
    PROD_INSTANCE p;
    ATTR_NO a;
{
    ifdef SDE_DEBUG_1
    /* debugging */
    printf("Entering met_error_store_delete\n");
    printf("Leaving met_error_store_delete\n");
    endif
}

/*------*/

met_error_store_insert(p, a, v)
    PROD_INSTANCE p;
    ATTR_NO a;
    PROD_INSTANCE v;
{
    HeadPtr
    h;
    Find_Header_Node_from_Op_or_Tmpl_or_TempSpec();
    void
    Move_To_Structure_Front();
    ifdef SDE_DEBUG_1
    /* debugging */
    printf("Entering met_error_store_insert\n");
    endif
    if (v == NULLVALUE) return;
    ifdef SDE_DEBUG_1
    /* debugging */
    printf("Leaving met_error_store_insert\n");
    endif
}
```

/*
 * met_error_store_insert: Op Name = %s
 * str2_to_str_ro( StrValue( Get_lvl( id_from_Op(p) ) ) )
 * printf("Met_error_store_insert: Met_error = %s\n", (BoolValue(v)) ? "true" : "false");
 * endif

 h = Find_Header_Name_from_Op_or_Tmppl_or_TempSpec[p];
 if (h != NULL)
 {
    h->met_error = BoolValue(v);
    /* enforce most-recently-used-first rule */
    if (prototype = h)
    {
        /* move header_node to front of list */
        Move_To_Structure_Front(h);
    }
    else
    {
        printf("met_error_store_insert: operator not found\n");
    }
}

#endif
```
APPENDIX D - Auxiliary Functions

PROD_INSTANCE v;

HeadPtr h,
Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

void
Move_To_Structure_Front();

#ifdef SDE_DEBUG_1
/* debugging */
printf("Entering inh_met_store_insert\n");
#endif

if (v == NULLVALUE) return;

h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

if (h != NULL)
{
  h->inh_met = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_to_Structure_Front(h);
  }
  else
  {
    printf("inh_met_store_insert: operator not found\n");
  }
#endif SDE_DEBUG_1

printf("Leaving inh_met_store_insert\n");
#endif SDE_DEBUG_1

*/
impl_store_init()

#ifdef SDE_DEBUG_1
/* debugging */
printf("Entering impl_store_init\n");
printf("Leaving impl_store_init\n");
#endif SDE_DEBUG_1

} /*---------------------------------------------------------*/

PROD_INSTANCE impl_store_fetch()

#ifdef SDE_DEBUG_1
/* debugging */
printf("Entering impl_store_fetch\n");
printf("Leaving impl_store_fetch\n");
#endif SDE_DEBUG_1

} /*---------------------------------------------*/

impl_store_delete(p, a)
PROD_INSTANCE p,
ATTR_NO a;

#ifdef SDE_DEBUG_1
/* debugging */
printf("Entering impl_store_delete\n");
printf("Leaving impl_store_delete\n");
#endif SDE_DEBUG_1

*/

---

217
APPENDIX D - Auxiliary Functions

/* vertex_ids_store_init() */
void
{
    #ifdef SDE_DEBUG_1
        /* debugging */
        printf("Entering vertex_ids_store_init\n");
        printf("Leaving vertex_ids_store_init\n");
    #endif

    /*
    */
    
    /* PROD_INSTANCE vertex_ids_store_fetch() */
    void
    {
        #ifdef SDE_DEBUG_1
            /* debugging */
            printf("Entering vertex_ids_store_fetch\n");
            printf("Leaving vertex_ids_store_fetch\n");
        #endif
    }

    /*
    */
    
    /* vertex_ids_store_delete(p, a) */
    PROD_INSTANCE p;
    ATTR_NO a;

    int
    {
        #ifdef SDE_DEBUG_1
            /* debugging */
            printf("Entering vertex_ids_store_delete\n");
            printf("Leaving vertex_ids_store_delete\n");
        #endif
    }

    /* vertex_ids_store_insert(p, a, v) */
    PROD_INSTANCE p;
    ATTR_NO a;
    PROD_INSTANCE v;

    (HeadPtr
    h,
    Find_Header_Node_from.Op.Or_TopImpl_or_TopSpec();
    void
    Move_To_Structure_Front();

    #ifdef SDE_DEBUG_1
        /* debugging */
        printf("Entering vertex_ids_store_insert\n");
    #endif

    if (v == NULLVALUE) return;

    #ifdef SDE_DEBUG_1
        /* debugging */
        printf("Vertex_ids_insert: Op_Name = \n",
            strToStr_ro(Stride(vec_Id_from.Op(p))));
    #endif

    h = Find_Header_Node_from.Op.Or_TopImpl_or_TopSpec(p);

    if (h != NULL)
    { h->vertex_id_set = v;

    /*
    */
    
    /* PROD_INSTANCE edge_ids_store_fetch() */
    
    void
    {
        // enforce most-recently-used-first rule */
        if (prototype != h)
        {
            /* move header_node to front of list */
            Move_To_Structure_Front(h);
        }
        else
        {
            printf("vertex_ids_store_insert: operator not found\n");
        }
    }

    /*
    */
    
    /* PROD_INSTANCE edge_ids_store_delete() */
    
    void
    {
        #ifdef SDE_DEBUG_1
            /* debugging */
            printf("Entering edge_ids_store_delete\n");
            printf("Leaving edge_ids_store_delete\n");
        #endif
    }

    /*
    */
    
    /* edge_ids_store_insert(p, a, v) */
    PROD_INSTANCE p;
    ATTR_NO a;
    PROD_INSTANCE v;

    (HeadPtr
    h,
    Find_Header_Node_from.Op.Or_TopImpl_or_TopSpec();
    void
    Move_To_Structure_Front();

    #ifdef SDE_DEBUG_1
        /* debugging */
        printf("Entering edge_ids_store_insert\n");
        printf("Leaving edge_ids_store_insert\n");
    #endif

    */
APPENDIX D - Auxiliary Functions

```c
#ifdef SDE_DEBUG_1
    /* debugging */
    printf("Entering edge_ids_store_insert\n");
#endif

    if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
    /* debugging */
    printf("Entering edge_ids_store_insert: Op Name = %s\n","str_to_str_ro(StrValue(Get_Id(id_from_Op(p))));
#endif

    h = Find_Header_Node_from_Op_or_TOPImpl_or_TOPSpec(p);
    if (h != NULL)
    {
        h->edge_id_set = v;
        /* enforce most-recently-used-first rule */
        if (prototype != h)
        {
            /* move header_node to front of list */
            Move_To_Structure_Front(h);
        }
        else
        {
            printf("edge_ids_store_insert: operator not found\n");
        }
    }
#endif

    /*----------------------------*/

#endif CANCEL_TEMP_DELETE

stream_ids_store_init()
{
    /* debuggin */
    printf("Entering stream_ids_store_init\n");
    printf("Leaving stream_ids_store_init\n");
#endif

    /*----------------------------*/

#pragma INSTANCE stream_ids_store_fetch()
{
    ifdef SDE_DEBUG_1
        /* debugging */
        printf("Entering stream_ids_store_fetch\n");
        printf("Leaving stream_ids_store_fetch\n");
    endif
}

    /*----------------------------*/

stream_ids_store_delete(p, a)
    attr_instance p;
    attr_instance a;
```
APPENDIX D - Auxiliary Functions

/ * debugging */
 printf("Entering constraints_ids_store_init\n");
 printf("Leaving constraints_ids_store_init\n");

 ifdef SDE_DEBUG_1
 /* debugging */
 printf("Entering constraints_ids_store_fetch\n");
 printf("Leaving constraints_ids_store_fetch\n");
 endif

 constraint_ids_store_delete(p, a)
 PROD_INSTANCE p;
 ATTRNO a;

 ifdef SDE_DEBUG_1
 /* debugging */
 printf("Entering constraints_ids_store_delete\n");
 printf("Leaving constraints_ids_store_delete\n");
 endif

 constraint_ids_store_insert(p, a, v)
 PROD_INSTANCE p;
 ATTRNO a;
 PROD_INSTANCE v;

 HeadPtr
 h,
 Find_Header_Node_from_Op_or_TopImpl_or_TopSpec();
 void
 Move_To_Structure_Front();

 ifdef SDE_DEBUG_1
 /* debugging */
 printf("Entering constraints_ids_store_insert\n");
 endif

 if (v == NULLVALUE) return;

 ifdef SDE_DEBUG_1
 /* debugging */
 printf("Constraints_ids_insert: Op Name = %sn, 
 str0_to_str(ro(Stro(Stro(Get_Id(id_from_Op(p))))));
 endif

 h = Find_Header_Node_from_Op_or_TopImpl_or_TopSpec(p);
 if (h != NULL)
 {
 h->constraints_id_set = v;
 /* enforce most-recently-used-first rule */
 if [prototype != h]
 {

 ifdef SDE_DEBUG_1
 /* move header_node to front of list */
 Move_To_Structure_Front(h);
 }
 else
 {
 printf("constraints_ids_store_insert: operator not found\n");
 }

 ifdef SDE_DEBUG_1
 printf("Leaving constraints_ids_store_insert\n");
 endif

 endif
 /*CANCEL_TEMP_DELETE*/

 multi_op_error_store_init()

 ifdef SDE_DEBUG_1
 /* debugging */
 printf("Entering multi_op_error_store_init\n");
 printf("Leaving multi_op_error_store_init\n");
 endif

 /*----------------------------------------------------------*/
 PROD_INSTANCE multi_op_error_store_fetch()

 ifdef SDE_DEBUG_1
 /* debugging */
 printf("Entering multi_op_error_store_fetch\n");
 printf("Leaving multi_op_error_store_fetch\n");
 endif

 /*----------------------------------------------------------*/
 multi_op_error_store_delete(p, a)
 PROD_INSTANCE p;
 ATTRNO a;

 ifdef SDE_DEBUG_1
 /* debugging */
 printf("Entering multi_op_error_store_delete\n");
 printf("Leaving multi_op_error_store_delete\n");
 endif

 /*----------------------------------------------------------*/
 multi_op_error_store_insert(p, a, v)
 PROD_INSTANCE p;
 ATTRNO a;
 PROD_INSTANCE v;

 HeadPtr
 h,
 Find_Header_Node_from_Op_or_TopImpl_or_TopSpec();
 void
 Move_To_Structure_Front();

 ifdef SDE_DEBUG_1
APPENDIX D - Auxiliary Functions

```c
/* debugging */
print("Entering multi_op_error_store_insert\n");
#endif

if (v == NULLVALUE) return;

#define SDE_DEBUG1
/* debugging */
print("Multi_op_error_store_insert: Op Name = %s\n", str_to_str_ro(StrValue(Get_ID(id_from_op(p)))));
print("Multi_op_error_store_insert: Multi_op_error = %s\n", BoolValue(v)) ? "true" : "false");
#endif

h = Find_Header_Node_from_op_or_TopImpl_or_TSpec(p);
if (h != NULL) {
    h->multi_op_error = BoolValue(v);
    /* enforce most-recently-used-first rule */
    if (prototype != h)
        /* move header node to front of list */
        Move_To_Structure_Front(h);
    } else
        print("multi_op_store_insert: operator not found\n");
#endif

#define SDE_DEBUG1
print("Entering multi_op_error_store_insert\n");
#endif

stream_error_store_init()
{
    #ifdef SDE_DEBUG1
    /* debugging */
    print("Entering stream_error_store_init\n");
    print("Leaving stream_error_store_init\n");
    #endif
    
    #ifdef SDE_DEBUG1
    /* debugging */
    print("Entering stream_error_store_fetch\n");
    print("Leaving stream_error_store_fetch\n");
    #endif
    
    #ifdef SDE_DEBUG1
    /* debugging */
    print("Entering stream_error_store_delete\n");
    print("Leaving stream_error_store_delete\n");
    #endif
    
    stream_error_store_delete(p, a);
    #endif

    stream_error_store_init(p, a, v)
    PROD_INSTANCE p;
    ATTR_NO a;
    PROD_INSTANCE v;
    {
        HeadPtr
        h,
        Find_Header_Node_from_op_or_TopImpl_or_TSpec(p);
        void
        Move_To_Structure_Front();
    
    #ifdef SDE_DEBUG1
    /* debugging */
    print("Entering stream_error_store_insert\n");
    #endif
    
    if (v == NULLVALUE) return;
    
    #ifdef SDE_DEBUG1
    /* debugging */
    print("Entering stream_error_store_insert\n");
    #endif
    
    h = Find_Header_Node_from_op_or_TopImpl_or_TSpec(p);
    if (h != NULL) {
        h->stream_error = BoolValue(v);
        /* enforce most-recently-used-first rule */
        if (prototype != h)
            /* move header node to front of list */
            Move_To_Structure_Front(h);
        } else
            print("stream_error_store_insert: Stream_error not found\n");
    
    #ifdef SDE_DEBUG1
    /* debugging */
    print("Entering stream_error_store_fetch\n");
    print("Leaving stream_error_store_fetch\n");
    #endif
    
    #ifdef SDE_DEBUG1
    /* debugging */
    print("Entering stream_error_store_init\n");
    print("Leaving stream_error_store_init\n");
    #endif
    
    #ifdef SDE_DEBUG1
    /* debugging */
    print("Entering constraint_error_store_init\n");
    print("Leaving constraint_error_store_init\n");
    #endif
    
    221
```
APPENDIX D - Auxiliary Functions

```c
/* */

/* */
PROD_INSTANCE constraint_error_store_fetch()
{
    #ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering constraint_error_store_fetch\n");
    printf("Leaving constraint_error_store_fetch\n");
    #endif

    }/* */

PROD_INSTANCE constraint_error_store_delete(p, a)
{
    #ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering constraint_error_store_delete\n");
    printf("Leaving constraint_error_store_delete\n");
    #endif

    }/* */

PROD_INSTANCE constraint_error_store_insert(p, a, v)
{
    #ifdef SDE_DEBUG1
    /* debugging */
    printf("Entering constraint_error_store_insert\n");
    #endif

    if (v == NULLVALUE) return;

    #ifdef SDE_DEBUG1
    /* debugging */
    printf("Constraint_error_store_insert: Op Name = %s\n",
            str0_to_str_cStrValue(Get_Id(id_from_OP(p))));
    printf("Constraint_error_store_insert: Constraint_error = %s\n",
            BoolValue(v)?"true":"false");
    #endif

    h = Find_Header_Node_from_Unicode_CPtr(p);
    if (h == NULL)
    {
        h = constraint_error = BoolValue(v);
    }/* enforce most-recently-used-first rule */
    // (prototype != h)
    { /* move header_node to front of list */
        Move_To_Structure_Front(h);
    }
}
```
APPENDIX D - Auxiliary Functions

```c
#ifndef SDE_DEBUG_1
/* debugging */
    printf("Entering t_multi_op_error_store_insert\n");
#endif

if (v == NULLVALUE) return;

#ifndef SDE_DEBUG_1
/* debugging */
    printf("Entering t_type_id_store_delete\n");
#endif

str0_to_str_ro(GetValue(Get_Id(id_from_TOpSpec(p))));
    printf("Multi_op_error = %s\n", (BoolValue(v)) ? "true": "false");
#endif

h = Find_Header_Node_from.Op_or.TOpImpl_or.TOpSpec(p);

if (h != NULL)
    /* enforce most-recently-used-first rule */
else
    /* need to make new header node */
    new_name = str0_to_str_ro(GetValue(Get_Id(id_from_TOpSpec(p))));
    prod_no = (int)p;
    h = Make.Operator.Header(new_name, NULL, NULL,
        GetUniqueId(), prod_no, FALSE);
    Link_To_Structure(h);
    h->multi_op_error = BoolValue(v);
#endif

#ifndef SDE_DEBUG_1
    printf("Leaving t_multi_op_error_store_insert\n");
#endif

/*-------------------------*/

/*-------------------------*/

#ifndef SDE_DEBUG_1
    printf("Leaving t_type_id_store_init\n");
#endif

/*-------------------------*/

/*-------------------------*/

/*-------------------------*/

PROD_INSTANCE t_type_id_store_fetch()
{
    /* not used here */
#ifndef SDE_DEBUG_1
    printf("Entering t_type_id_store_fetch\n");
#endif
    printf("Leaving t_type_id_store_fetch\n");
#endif

/*-------------------------*/

/*-------------------------*/

if (v == NULLVALUE) return;

#ifndef SDE_DEBUG_1
    printf("Entering t_type_id_store_delete\n");
#endif

str0_to_str_ro(GetValue(Get_Id(id_from_TOpSpec(p))));
    printf("Leaving t_type_id_store_delete\n");
#endif

/*-------------------------*/

t_type_id_store_insert(p, a, v)
PROD_INSTANCE p;
    ATTR_NO a;
    PROD_INSTANCE v;

    int prod_no;
    char *new_type_name;
    HeadPtr h;
    Make.Operator.Header();
    Find.Header.Node_from.Op_or.TOpImpl_or.TOpSpec();
    PROD_INSTANCE t;
    void Move_To_Structure_Front(),
    Link.To.Structure();
#endif

#ifndef SDE_DEBUG_1
    printf("Entering t_type_id_store_init\n");
#endif

#ifndef SDE_DEBUG_1
    printf("Entering t_type_id_store_init\n");
#endif

if (h != NULL)
    {if (strcmp(h->type_id, ") != 0]
        free(h->type_id);
        h->type_id = strdup(str0_to_str_ro(GetValue(Get_Id(v))));
        /* enforce most-recently-used-first rule */
if (prototype != h)
{ /* move header node to front of list */
    Move_To_Structure_Front(h);
}
else
{
    printf("_type_id_store_insert: operator not found\n");
}
#endif // SDE_DEBUG_1

#define _type_id_store_insert

#endif // SDE_DEBUG_1

#define _type_id_store_insert

#define _type_id_store_insert

#define _type_id_store_delete

#define _type_id_store_delete

#define _type_id_store_delete

#define _type_id_store_delete

#define _type_id_store_delete

#define _type_id_store_delete
APPENDIX D - Auxiliary Functions

/*
 */
if (v == NULLVALUE) return;
#endif DEBUG_1
    /* debugging */
    printf("Operator_id_store_insert: Op Name = %sn",
            str0_to_str_ro(StrValue(Get_Id(id_from_TCopImpl[p]))));
#endif

h = Find_Header_Node_from_Op_or_TCopImpl_or_TCopSpec(p);

prod_no = (int)p;
new_name = str0_to_str_ro(StrValue(Get_Id(v)));
#endif DEBUG_2
    /* debugging */
    printf("INSERT Operator is %d\n", new_name, prod_no);
#endif

if (h == NULL)
    /* need to make new header node */
    h = Make_0perator_Hd(new_name, NULL, NULL,
                        Get_Uname_Id(), prod_no, FALSE);
else
    /* operator already in the list, update name and prod_no */
    free(h->name);
    free(h->ada_op_name);
    h->name = strdup(new_name);
    h->ada_op_name = strdup(new_name);
    h->prod_no = h->prod_no;
#endif DEBUG_1
    /* enforce most-recently-used-first rule */
    if (prototype != h)
        /* move header node to front of list */
        Move_To_Structure_Front(h);
#endif DEBUG_1
    printf("Leaving t_operate_id_store_insert\n");
#endif DEBUG_1

/*
 */
t_states_id_store_init()
{
#endif DEBUG_1
    /* debugging */
    printf("Entering t_states_id_store_init\n");
    printf("Leaving t_states_id_store_init\n");
#endif
}

/*
 */
t_states_id_store_delete(p, a)
{
    PROD_INSTANCE p;
    ATTR_NO a;
    #ifndef DEBUG_1
    /* debugging */
    printf("Entering t_states_id_store_delete\n");
    printf("Leaving t_states_id_store_delete\n");
    #endif
}

/*
 */
t_states_id_store_insert(p, a, v)
{
    PROD_INSTANCE p;
    ATTR_NO a;
    PROD_INSTANCE v;
    HeadPtr h,
    Find_Header_Node_from_Op_or_TCopImpl_or_TCopSpec();
    void
    Move_To_Structure_Front();
#endif DEBUG_1
    /* debugging */
    printf("Entering t_states_id_store_insert\n");
    #endif

if (v == NULLVALUE) return;
#endif DEBUG_1
    /* debugging */
    printf("t_states_id_store_insert: Op Name = %d\n",
            str0_to_str_ro(StrValue(Get_Id(id_from_TCopImpl[p]))));
#endif

h = Find_Header_Node_from_Op_or_TCopImpl_or_TCopSpec(p);
if (h != NULL)
    h->state_id_set = v;
#endif DEBUG_1
    /* enforce most-recently-used-first rule */
    if (prototype != h)
        /* move header node to front of list */
        Move_To_Structure_Front(h);
#else DEBUG_1
    printf("t_states_id_store_insert: operator not found\n");
#endif
}
APPENDIX D - Auxiliary Functions

```c
#ifndef SDE_DEBUG_1

#endif

// --------------------

/* t_inh_input_ids_store_init() */
#endif

/* t_inh_input_ids_store_fetch() */
#endif

/* t_inh_input_ids_store_delete(p, a) */
PROD_INSTANCE p;
ATTR_NO a;
#endif

/* t_inh_input_ids_store_insert(p, a, v) */
PROD_INSTANCE p;
ATTR_NO a;
PROD_INSTANCE v;

HeadPtr h,
Find_Header_Node_from_Op_or_TopImpl_or_TOpSpec();
void
Move_To_Structure_Front();

#ifndef SDE_DEBUG_1
/* debugging */
printf("Entering t_inh_input_ids_store_insert\n");
#endif

if (v == NULLVALUE) return;
#endif

/* t_inh_input_ids_store_insert: Op Name = %s
str0_to_str_ro(StrValue(Get_Id(id_from_TopImpl[p]))); */
#endif

h = Find_Header_Node_from_Op_or_TopImpl_or_TOpSpec(p);
if (h != NULL)
{
    h->inh_input_id_set = v;
    /* enforce most-recently-used-first rule */
    if (prototype != h)
    {
        /* move header_node to front of list */
        Move_To_Structure_Front(h);
    }
    else
    {
        printf("t_inh_input_ids_store_insert: operator not found\n");
    }
#endif

/* t_inh_output_ids_store_set() */
#endif

/* t_inh_output_ids_store_fetch() */
#endif

/* t_inh_output_ids_store_delete(p, a) */
PROD_INSTANCE p;
ATTR_NO a;
#endif

/* t_inh_output_ids_store_insert(p, a, v) */
PROD_INSTANCE p;
ATTR_NO a;
PROD_INSTANCE v;

HeadPtr h,
Find_Header_Node_from_Op_or_TopImpl_or_TOpSpec();
```

226
APPENDIX D - Auxiliary Functions

```c
void
Move_To_Structure_Front();

#define SDE_DEBUG1
/* debugging */
printf("Entering t_inh_output_ids_store_insert\n");
#endif

if (v == NULLVALUE) return;

#define SDE_DEBUG1
/* debugging */
printf("Entering t_inh_output_ids_store_insert; Op Name = %s\n", strToStr_ro(Stride(Get_Id(id_from_TOpImp1(p)))));
#endif

h = Find_Header_Node_from_Op_or_TOpImp1_or_TOpSpec(p);
if (h != NULL)
{
    h->inh_output_id_set = v;
    /* enforce most-recently-used-first rule */
    if (prototype != h)
    {
        /* move header_node to front of list */
        Move_To_Structure_Front(h);
    }
}
else
{
    printf("t_inh_output_ids_store_insert; operator not found\n");
}
#endif

#define SDE_DEBUG1
printf("Leaving t_inh_output_ids_store_insert\n");
#endif

/*-----------------------------*/
t_inh_met_store_init()
{
#define SDE_DEBUG1
/* debugging */
printf("Entering t_inh_met_store_init\n");
#endif

/*-----------------------------*/
PROD_INSTANCE t_inh_met_store_fetch()
{
#define SDE_DEBUG1
/* debugging */
printf("Entering t_inh_met_store_fetch\n");
#endif

/*-----------------------------*/
t_inh_met_store_delete(p, a)
PROD_INSTANCE p;

#define SDE_DEBUG1
/* debugging */
printf("Entering t_inh_met_store_delete\n");
#endif

if (v == NULLVALUE) return;

#define SDE_DEBUG1
/* debugging */
printf("Entering t_inh_met_store_delete; Op Name = %s\n", strToStr_ro(Stride(Get_Id(id_from_TOpImp1(p)))));
#endif

h = Find_Header_Node_from_Op_or_TOpImp1_or_TOpSpec(p);
if (h != NULL)
{
    h->inh_met = v;
    /* enforce most-recently-used-first rule */
    if (prototype != h)
    {
        /* move header_node to front of list */
        Move_To_Structure_Front(h);
    }
}
else
{
    printf("t_inh_met_store_insert; operator not found\n");
}
#endif

#define SDE_DEBUG1
printf("Leaving t_inh_met_store_insert\n");
#endif

/*-----------------------------*/
t_timpl_store_init()
{
#define SDE_DEBUG1
/* debugging */
printf("Entering t_timpl_store_init\n");
#endif

if (v == NULLVALUE) return;

#define SDE_DEBUG1
/* debugging */
printf("Entering t_timpl_store_init; Op Name = %s\n", strToStr_ro(Stride(Get_Id(id_from_TOpImp1(p)))));
#endif

h = Find_Header_Node_from_Op_or_TOpImp1_or_TOpSpec(p);
if (h != NULL)
{
    h->timpl = v;
    /* enforce most-recently-used-first rule */
    if (prototype != h)
    {
        /* move header_node to front of list */
        Move_To_Structure_Front(h);
    }
}
else
{
    printf("t_timpl_store_insert; operator not found\n");
}
#endif

#define SDE_DEBUG1
printf("Leaving t_timpl_store_insert\n");
#endif
```
APPENDIX D - Auxiliary Functions

```c
/* debugging */
printf("Entering t_impl_store_init\n");
printf("Leaving t_impl_store_init\n");
#endif
}
/*-----------------------------------------------*/
PROD_INSTANCE t_impl_store_fetch()
{
    ifndef SDE_DEBUG_1
    /* debugging */
    printf("Entering t_impl_store_fetch\n");
    printf("Leaving t_impl_store_fetch\n");
    #endif
}

/*-----------------------------------------------*/
t_impl_store_delete(p, a)
PROD_INSTANCE p;
ATTR_NO a;
{
    ifndef SDE_DEBUG_1
    /* debugging */
    printf("Entering t_impl_store_delete\n");
    printf("Leaving t_impl_store_delete\n");
    #endif
}

/*-----------------------------------------------*/
t_impl_store_insert(p, a, v)
PROD_INSTANCE p;
ATTR_NO a;
PROD_INSTANCE v;
{
    PROD_INSTANCE
ValidOp_Impl();

    HeadPtr h,
Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

    void
Move_To_Structure_Front();
#endif SDE_DEBUG_1
/* debugging */
printf("Entering t_impl_store_insert\n");
#endif
if (v == NULLVALUE) return;
#endif SDE_DEBUG_1
/* debugging */
printf("Entering t_impl_store_insert\nOp Name = %s\n",
str0_to_str_cstr(StrValue(Get_Id[id_from_TOpImpl[p]][]]));
#endif
h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);
if (h != NULL)
{
    if (IntValue(v) < 2)
        h->is_composite = false;

    else
        h->is_composite = true;

    */ enforce most-recently-used-first rule */
    if (prototype != h)
    { /* move header_node to front of list */
        Move_To_Structure_Front(h);
    }
    else
    {
        printf("t_impl_store_insert: operator not found\n");
    }
#endif SDE_DEBUG_1
printf("Leaving t_impl_store_insert\n");
#endif
}

/*-----------------------------------------------*/
t_vertex_ids_store_init()
{
    ifndef SDE_DEBUG_1 /* debugging */
    printf("Entering t_vertex_ids_store_init\n");
    printf("Leaving t_vertex_ids_store_init\n");
    #endif
}

/*-----------------------------------------------*/
t_vertex_ids_store_fetch()
{
    #ifndef SDE_DEBUG_1
    /* debugging */
    printf("Entering t_vertex_ids_store_fetch\n");
    printf("Leaving t_vertex_ids_store_fetch\n");
    #endif
}

/*-----------------------------------------------*/
t_vertex_ids_store_delete(p, a)
PROD_INSTANCE p;
ATTR_NO a;
{
    ifndef SDE_DEBUG_1
    /* debugging */
    printf("Entering t_vertex_ids_store_delete\n");
    printf("Leaving t_vertex_ids_store_delete\n");
    #endif
}

/*-----------------------------------------------*/
t_vertex_ids_store_insert(p, a, v)
PROD_INSTANCE p;
ATTR_NO a;
PROD_INSTANCE v;
{
    HeadPtr h,
Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

    void
Move_To_Structure_Front();
```
APPENDIX D - Auxiliary Functions

```c
#define SDE_DEBUG_1
/* debugging */
printf("Entering t_vertex_ids_store_insert\n");
#endif

if (v == NULLVALUE) return;

#define SDE_DEBUG_1
/* debugging */
printf("t_vertex_ids_insert: Op Name = %s\n", str_to_str_ro(StValue(Get_Id(t_id_from_TcpImpl(p))));
#endif

h = Find_Header_Node_from_TcpImpl_or_TcpSpec(p);
if (h != NULL)
    h->vertex_id_set = v;
/* enforce most-recently-used-first rule */
if (prototype != h)
    /* move header node to front of list */
    Move_To_Structure_Front(h);
#else
    printf("t_vertex_ids_store_insert: operator not found\n");
#endif

#define SDE_DEBUG_1
printf("Leaving t_vertex_ids_store_insert\n");
#endif

/*----------------------------------------*/
t_edge_ids_store_init()
{
#endif SDE_DEBUG_1
/* debugging */
printf("Entering t_edge_ids_store_init\n");
#endif

.getOrElse

/*----------------------------------------*/
PROD_INSTANCE t_edge_ids_store_fetch()
{
#endif SDE_DEBUG_1
/* debugging */
printf("Entering t_edge_ids_store_fetch\n");
#endif

.getOrElse

/*----------------------------------------*/
t_edge_ids_store_delete(p, a)
    PROD_INSTANCE p;
    ATTR_No a;
#endif SDE_DEBUG_1
/* debugging */
printf("Entering t_edge_ids_store_delete\n");
#endif

.getOrElse

/*----------------------------------------*/
t_stream_error_store_init()
{
#endif SDE_DEBUG_1
/* debugging */
printf("Entering t_stream_error_store_init\n");
#endif

.getOrElse
```
APPENDIX D - Auxiliary Functions

```c
#endif

/*-------------------------------------------------------------*/
PROD_INSTANCE t_stream_error_store_fetch()
{
    #ifdef SDE_DEBUG
    /* debugging */
    printf("Entering t_stream_error_store_fetch\n");
    printf("Leaving t_stream_error_store_fetch\n");
    #endif

    /*-------------------------------------------------------------*/
    t_stream_error_store_delete(p, a)
    PROD_INSTANCE p;
    ATTR_NO a;

    #ifdef SDE_DEBUG
    /* debugging */
    printf("Entering t_stream_error_store_delete\n");
    printf("Leaving t_stream_error_store_delete\n");
    #endif

    /*-------------------------------------------------------------*/
    t_stream_error_store_insert(p, a, v)
    PROD_INSTANCE p;
    ATTR_NO a;
    PROD_INSTANCE v;

    HeadPtr
    h;
    Find_Header_Node_from_Op_or_TopImpl_or_TopSpec();

    void
    Move_To_Structure_Front();

    #ifdef SDE_DEBUG
    /* debugging */
    printf("Entering t_stream_error_store_insert\n");
    #endif

    if (v == NULLVALUE) return;

    #ifdef SDE_DEBUG
    /* debugging */
    printf("T_stream_error_store_insert: Op Name = %s\n", str0_to_str_ro(StrValue(Get_Id(id_from_TopImpl(p)))));
    printf("T_stream_error_store_insert: Stream_error = %s\n", BoolValue(v) ? "true", "false");
    #endif

    h = Find_Header_Node_from_Op_or_TopImpl_or_TopSpec(p);
    if (h != NULL)
    {
        h->stream_error = BoolValue(v);
        /* enforce most-recently-used-first rule */
        if (prototype != h)
        {
            /* move header_node to front of list */
    Move_To_Structure_Front(h);
        }
    } else
    {
        printf("t_stream_error_store_insert: operator not found\n");
    }

    #ifdef SDE_DEBUG
    printf("Leaving t_stream_error_store_insert\n");
    #endif

    /*-------------------------------------------------------------*/
    t_constraint_error_store_init()
    {
    #ifdef SDE_DEBUG
    /* debugging */
    printf("Entering t_constraint_error_store_init\n");
    printf("Leaving t_constraint_error_store_init\n");
    #endif

    /*-------------------------------------------------------------*/
    t_constraint_error_store_fetch()
    {
    #ifdef SDE_DEBUG
    /* debugging */
    printf("Entering t_constraint_error_store_fetch\n");
    printf("Leaving t_constraint_error_store_fetch\n");
    #endif

    /*-------------------------------------------------------------*/
    t_constraint_error_store_delete(p, a)
    PROD_INSTANCE p;
    ATTR_NO a;

    #ifdef SDE_DEBUG
    /* debugging */
    printf("Entering t_constraint_error_store_delete\n");
    printf("Leaving t_constraint_error_store_delete\n");
    #endif

    /*-------------------------------------------------------------*/
    t_constraint_error_store_insert(p, a, v)
    PROD_INSTANCE p;
    ATTR_NO a;
    PROD_INSTANCE v;

    HeadPtr
    h;
    Find_Header_Node_from_Op_or_TopImpl_or_TopSpec();

    void
    Move_To_Structure_Front();

    #ifdef SDE_DEBUG
    /* debugging */
    printf("Entering t_constraint_error_store_insert\n");
    #endif

    if (v == NULLVALUE) return;
```
APPENDIX D - Auxiliary Functions

```c
#ifdef SDR_DEBUG_1
    /* debugging */
    printf("t_constraint_error_store_insert: Cn Name = %s\n", str_to_str_ro(StrValue(Get_Id(id_from_topImpl(p)))));
    printf("t_constraint_error_store_insert: Constraint_error = %s\n", (BoolValue(v)) ? "true" : "false");
#endif

/* p is a t_op_impl production */
h = Find_Header_Node_from_Cp_or_TopImpl_or_TopSpec(p);
if (h != NULL)
{
    h->constraint_error = BoolValue(v);
    /* enforce most-recently-used-first rule */
    if (prototype != h)
    {
        /* move header_node to front of list */
        Move_ToStructure_Front(h);
    }
    else
    {
        printf("t_constraint_error_store_insert: operator not found\n");
    }
#endif

#ifdef SDR_DEBUG_1
    printf("Leaving t_constraint_error_store_insert\n");
#endif

/* ---------------------------------------- */
t_undefined_op_impl_store_init()
{
    /* debugging */
    printf("Entering t_undefined_op_impl_store_init\n");
    printf("Leaving t_undefined_op_impl_store_init\n");
#endif

/* ---------------------------------------- */
PROD_INSTANCE t_undefined_op_impl_store_fetch()
{
    /* debugging */
    printf("Entering t_undefined_op_impl_store_fetch\n");
    printf("Leaving t_undefined_op_impl_store_fetch\n");
#endif

/* ---------------------------------------- */
t_undefined_op_impl_store_delete(p, a)
    PROD_INSTANCE p;
    ATTR_NO a;
{
    /* debugging */
    printf("Entering t_undefined_op_impl_store_delete\n");
    printf("Leaving t_undefined_op_impl_store_delete\n");
#endif

/* ---------------------------------------- */
t_obsolete_op_impl_store_insert(p, a, v)
    PROD_INSTANCE p;
    ATTR_NO a;
    PROD_INSTANCE v;
{
    PROD_INSTANCE
    Get_Id();
    TYPE_LIST
t_i,
    Find_Type_Node_from_Data();
    Make_Type_Node();
#endif

    if (v == NULLVALUE) return;

t_i = Find_Type_Node_from_Data(p);
    if (t_i == NULL)
    t_i = Make_Type_Node(str_to_str_ro(StrValue(Get_Id(id_from_data(p)))));
    t_i->undefined_op_impl = v;
#endif

#ifdef SDR_DEBUG_1
    printf("Entering t_undefined_op_impl_store_insert\n");
#endif

/* ---------------------------------------- */
t_obsolete_op_impl_store_init()
{
    /* debugging */
    printf("Entering t_obsolete_op_impl_store_init\n");
    printf("Leaving t_obsolete_op_impl_store_init\n");
#endif

/* ---------------------------------------- */
PROD_INSTANCE t_obsolete_op_impl_store_fetch()
{
    /* debugging */
    printf("Entering t_obsolete_op_impl_store_fetch\n");
    printf("Leaving t_obsolete_op_impl_store_fetch\n");
#endif

/* ---------------------------------------- */
t_obsolete_op_impl_store_delete(p, a)
    PROD_INSTANCE p;
    ATTR_NO a;
{
    /* debugging */
    printf("Entering t_obsolete_op_impl_store_delete\n");
    printf("Leaving t_obsolete_op_impl_store_delete\n");
#endif
```

231
APPENDIX D - Auxiliary Functions

ASSWORD

/**
 * t_obsolete_op_impl_store_insert(p, a, v)
 * PROD_INSTANCE p;
 * ATRR_NO a;
 * PROD_INSTANCE v;
 */

PROD_INSTANCE
Get_Id();

TYPE_LIST
tl,
Find_Type_Node_from_Data(),
Make_Type_Node();

#define SDE_DEBUG_1
/* debugging */
printf("Entering t_obsolete_op_impl_store_insert\n");
#endif

if (v == NULLVALUE) return;

tl = Find_Type_Node_from_Data(p);
if (tl == NULL)

tl = Make_Type_Node(str0_to_str_ro(StringValue(Get_Id(id_from_Data[p]))));

tl->obsolete_op_impl = v;

#define SDE_DEBUG_1
/* debugging */
printf("leaving t_obsolete_op_impl_store_insert\n");
#endif

-------------

int RemoveHeadNode(pointed_at)

HeadPtr pointed_at;
{
    extern HeadPtr
    prototype;
    HeadPtr
    p,
    q

#define SDE_DEBUG_1
/* debugging */
printf("Entering RemoveHeadNode\n");
#endif

q = NULL;
p = prototype;
while (p != pointed_at)
{
    q = p;
p = p->next;
}

if (q == NULL)
{
    prototype = p->next;
}

else
{
    q->next = p->next;
}
free(p);

/*-----------------------*/
WipeOutSpline(s)

SPLINE_PTR s;

{ SPLINE_PTR last;

#if defined SDE_DEBUG_1
/* debugging */
printf("Entering WipeOutSpline\n");
#endif

while (s != NULL)
{
    last = s;
    s = s->next;
    free(last);
}

}/*-----------------------*/

WipeOutStreams(p)

STREAM_PTR p;

{ STREAM_PTR last;

#if defined SDE_DEBUG_1
/* debugging */
printf("Entering WipeOutStream\n");
#endif

while (p != NULL)
{
    last = p;
    q = p->st;
    WipeOutSpline(q->arc);
    free(q);
    p = p->next;
    free(last);
}

#if defined SDE_DEBUG_1
printf("WipeOutStreams\n");
#endif

}/*-----------------------*/

WipeOutOperators(p)

OPNodePTR p;

{ OPNodePTR last;

#if defined SDE_DEBUG_1
/* debugging */
printf("Entering WipeOutOperators\n");
#endif

while (p != NULL)
{...
APPENDIX D - Auxiliary Functions

```c
#include SDE_DEBUG_1

#define (*debug("Wipe our operators\n\>):

/**
** This eighth (last) set of functions were made in support of this thesis
** and was named functions2.asi

Application: CAPS/SDE (Syntax Directed Editor)
System: UNIX
Programmer: Scott Grenshooter
Last Updated: 951205
Purpose: Functions created in support of identifying simple
      timing and trigger constraints within a CAPS prototype
      so that simple errors can be identified and reported to
      the user while s/he is still in the editor, thereby
      reducing the time spent going back and forth from
      the scheduler to the editor.

* /

BOOL exported Is_Constrained_Operator(component c) {
    with(c) {
        NoComponent: false,
        Data(l, ts, tl): false,
        Op(i, os, ol):
            with(os) {
                OperatorSpec(*, '*', '*', '*', opt_net, '*', '*', '*'):
                    with(opt_met) {
                        OptTimingInfoNone: false,    
                        OptTimingInfoPrompt: false,  
                        OptTimingInfo(ol, '*'): true  
                    }
            }
        }
    }
}

/** Do the constraints belong to a Constrained Operator (has a NET)? */

BOOL exported Belong2Constrained_Operator(a_constraint ac, 
    op_id met_set id_met_set) {
    with(ac) {
        AConstraintNull: false,
        AConstraint(op_id, '*', '*', '*', '*', '*', '*', '*'):
            Is_OpID_In_IDMetSet(op_id, id_met_set) 
    }
    Is_OpID_In_IDMetSet(operator_id op_id, 
        op_id met_set id_met_set) {
            with(id_met_set) {
                OpIDMetSetNil: false,
                OpIDMetNull: Id_OpID_In_IDMetSet(op_id, tl),
                OpIDMet(id, '*'):
                    (op_id == id)
                        ? true
                        : Is_OpID_In_IDMetSet(op_id, tl)
            }
        }
    }
    per Get_Period(operator_id opid, id_constraint_set cs) {
        with(cs) {
            IdConstraintSetNil: PerNull,
            IdConstraintPair(hd, tl):
                with(hd) {
                    IdConstraintNull: Get_Period(opid, tl),
                    IdConstraint(id, tc, '*'):
                        (opid == id)
                        ? with(tc) {
                            OpConstraintNull: PerNull,
                            Periodic(per, '*'): per,
                            Sporadic(*, '*'): PerNull
                        }
                        : Get_Period(opid, tl)
            }
        }
    }

BOOL exported Is_ProducerOp_Period_LE_ConsumerOp(edge_set es, 
    id_met_set cs) {
    with(es) {
        EdgeSetNil: false,
        EdgePair(hd, tl):
            with(hd) {
                EdgeNull: Is_ProducerOp_Period_LE_ConsumerOp(tl, cs),
                EdgePair(';', p, c):
                    with(p) {
                        ProducerNull: Is_ProducerOp_Period_LE_ConsumerOp(tl, cs),
                        Producer(p, p_opid):
                            with(c) {
                                ConsumerNull: Is_ProducerOp_Period_LE_ConsumerOp(tl, cs),
                                Consumer(c, p_opid):
                                    (Get_Period(p, p_opid, cs) == PerNull ||
                                        Get.PERIOD(p, p_opid, cs) == PerNull) 
                                        : Get_Period(p, p_opid, cs) <= Get.PERIOD(c, cs)
                                        ? true
                                        : Is_ProducerOp_Period_LE_ConsumerOp(tl, cs)
                                    }
                            }
                        }
                    }
                }
            }
        }
    }
    */
```

233
with(hd) {  
  Is_Sporadic(opid, tl),
  idConstraintNull: opid,
}  
with(tc) {
  OpConstraintNull: false,
  Periodic(*, *): false,
  Sporadic(*, *): true
}  
  }  
  }  
  }  
  }  
}  

trigger Get_Trigger(operator_id opid, id_constraint_set cs) {  
with(cs) {  
  idConstraintSetNull: TriggerByNull,
  idConstraintFair(hd, tl),
}  
with(hd) {  
  idConstraintNull: Get_Trigger(opid, tl),
  idConstraintNull:, id: true
}  
}  
\hfill 

BOOL exported
Is_Sporadic_ConsumerOp_WO_Trigger(edge_set es, op_id_met_set ms, id_constraint_set cs) {
with(es) {  
  EdgeSetNull: false,
  EdgeFair(hd, tl),
}  
with(hd) {
  EdgeNull: Is_Sporadic_ConsumerOp_WO_Trigger(tl, cs),
  Edge(*, *, *, c):
}  
with(c) {  
  ConsumerNull: Is_Sporadic_ConsumerOp_WO_Trigger(tl, cs),
  Consumer(c_opid):  
    (Is_Sporadic(c_opid, cs))
}  
  }  
}  

BOOLO exported
Is_ConstrProducerOp_And_ConstrConsumerOp_W_Trigger(edge_set es, op_id_met_set ms, id_constraint_set cs) {  
with(es) {  
  EdgeSetNull: false,
  EdgeFair(hd, tl),
}  
with(hd) {  
  EdgeNull: Is_ConstrProducerOp_And_ConstrConsumerOp_W_Trigger(tl, ms, cs),
  Edge(*, *, p, c):
}  
with(p) {  
  ProducerNull: Is_ConstrProducerOp_And_ConstrConsumerOp_W_Trigger(tl, ms, cs),
  Producer(p_opid):  
    (Is_ConstrProducerOp_And_ConstrConsumerOp_W_Trigger(tl, ms, cs))
}  
}  

/\--------------------------*/

BOOL Is_Constrainted_Opid(operator_id opid, op_id_met_set ms) {  
with(ms) {  
  OpIdMetSetNull: false,
  OpIdMetFair(hd, tl),
}  
with(hd) {  
  OpIdMetNull: Is_Constrained_Opid(opid, tl),
  OpIdMet(opid, met),
  (opid == old)
}  
  }  
  }  
}  

234
APPENDIX D - Auxiliary Functions

BOOL Is_Ncp(optional_mcp o_mcp) {
    with(o_mcp) {
        OptMcpNull: false,
        OptMcpPrompt: true
    }
}

/* Returns true if the constraint passed in has a time value for any of its four timing constraints. */
BOOL exported Partial_Constraint(sa_constraint ac) {
    with(ac) {
        AConstraintNull: false,
        AConstraint(*, *, o_per, o_fw, o_mcp, o_mrt, *, *, *):
            Is_Per(o_per) || Is_FW(o_fw) ||
            Is_Mrt(o_mrt) || Is_Ncp(o_mcp)
    }
}

/* Returns true if constraints are Periodic XOR Sporadic. */
BOOL exported Valid_T_Constraint(sa_constraint ac) {
    with(ac) {
        AConstraintNull: false,
        AConstraint(*, *, o_per, o_fw, o_mcp, o_mrt, *, *, *):
            Partial_Constraint(ac) &&
            (Is_Per(o_per) || Is_FW(o_fw))
    }
}

/* Net <= FW < PER must exist for an Operator to be scheduleable. */
BOOL exported Unschedulable_Periodic_Op(sa_constraint ac, op_id, net_set, met_set) {
    with(ac) {
        AConstraintNull: false,
        AConstraint(op_id, *, o_per, o_fw, *, *, *, *):
            Is_Per(o_per) || Is_FW(o_fw)
    }
}

BOOL Is_Per(optional_period o_per) {
    with(o_per) {
        OptPeriodNull: false,
        OptPeriodPrompt: false,
        OptPeriod(true)
    }
}

BOOL Is_FW(optional_finish_within o_fw) {
    with(o_fw) {
        OptFinishWithinNull: false,
        OptFinishWithinPrompt: false,
        OptFinishWithin(true)
    }
}

BOOL Is_Mrt(optional_mrt o_mrt) {
    with(o_mrt) {
        OptMrtNull: false,
        OptMrtPrompt: true
    }
}

/* 2*Net <= MRT < 2*MCP must exist for an Operator to be scheduleable. */
BOOL exported Unschedulable_Sporadic_Op(a_constraint ac, op_id_met_set met_set) {
    with(ac) {
        AConstraintNull; false,
        AConstraint(op_id, '*', '*', o_mcp, o_mrt, '*', '*');
        { Is_Nrt(o_mrt) || Is_Mcp(o_mcp) }
        ? with(Get_Nrt(op_id, met_set)) {
            NrtNull; false,
            Nrt(met_num),
            with(Get_Mcp(o_mcp)) {
                McpNull; false,
                Mcp(met_num),
                MrtNull; false,
                Mrt(mrt_num),
                { 2.0 * met_num > mrt_num } ||
                { met_num > 2.0 * mcp_num }
            }
        }
    }
    ; false
}
}
BOOL exported PeriodOnly(a_constraint ac) {
    with(ac) {
        AConstraintNull; false,
        AConstraint('*', '*', o_per, o_fw, o_mcp, o_mrt, '*', '*');
        { Is_Per(o_per) && !Is_Per(o_fw) }
    }
}
BOOL exported FinishWithinOnly(a_constraint ac) {
    with(ac) {
        AConstraintNull; false,
        AConstraint('*', '*', o_per, o_fw, o_mcp, o_mrt, '*', '*');
        { !Is_Per(o_per) && Is_FW(o_fw) }
    }
}
BOOL exported MaxRespTimeOnly(a_constraint ac) {
    with(ac) {
        AConstraintNull; false,
        AConstraint('*', '*', o_per, o_fw, o_mcp, o_mrt, '*', '*');
        { Is_Nrt(o_mrt) && Is_Mcp(o_mcp) }
    }
}
BOOL exported MinCallPeriodOnly(a_constraint ac) {
    with(ac) {
        AConstraintNull; false,
        AConstraint('*', '*', o_per, o_fw, o_mcp, o_mrt, '*', '*');
        { !Is_Nrt(o_mrt) && Is_Mcp(o_mcp) }
    }
}
REAL Smallest_Per(id_constraint_set cs, REAL ans) {
    with(cs) {
        IdConstraintSetNull; ans,
        IdConstraintPair(hd, tl),
        with(hd) {
            IdConstraintNull; Smallest_Per(tl, ans),
            IdConstraint('*', tc, ' ');
            with(tc) {
                OpConstraintNull; Smallest_Per(tl, ans),
                Periodic(per, '*');
                with(per) {
                    PerNull; Smallest_Per(tl, ans),
                    Per(per_num) {
                        per_num < ans
                    } ? Smallest_Per(tl, per_num) :
                        Smallest_Per(tl, ans)
                };
                Sporadic(mcp, mrt); /* <<<<<<<<<<<<<< THIS IS NOT CORRECT. */
                Smallest_Per(tl, ans)
            }
        }
    }
    REAL Largest_Nrt(op_id_met_set ms, REAL ans) {
        with(ms) {
            OpIdMetSetNull; ans,
            OpIdMetPair(hd, tl),
            with(hd) {
                OpIdMetNull; Largest_Nrt(tl, ans),
                OpIdMet('*', m),
                with(m) {
                    MetNull; Largest_Nrt(tl, ans),
                    Met(met_num) {
                        met_num > ans
                    } ? Largest_Nrt(tl, met_num) :
                        Largest_Nrt(tl, ans)
                }
            }
        }
    }
    BOOL exported Uniprocessor_Schedulable(op_id_met_set ms, id_constraint_set cs) {
        Smallest_Per(cs, 10000000000000.0) > Largest_Nrt(ms, 0.0)
    }
    /* Does the actual work of getting # processors required. 
    Called by Min_Processors_Required which converts to int. */
    REAL Min_Processors_Required(op_id_met_set ms, id_constraint_set cs) {
        with(cs) {
            IdConstraintSetNull; 0.0,
            IdConstraintPair(hd, tl),
            with(hd) {
                IdConstraintNull; 0.0 + Min_Processors_Required(ms, tl),
                IdConstraint(old, tc, ' ');
                with(tc) {
                    OpConstraintNull; 0.0 + Min_Processors_Required(ms, tl),
                    Periodic(per, '*');
                    with(per) {
                        PerNull; 0.0 + Min_Processors_Required(ms, tl),
                        Per(per_num) {
                            per_num < old
                        } ? Min_Processors_Required(ms, per_num) :
                            Min_Processors_Required(ms, old)
                    }
                }
            }
        }
    }
}

236
APPENDIX D - Auxiliary Functions

```c
HEt(he_num) {
   (he_num / per_num) + min_processors_required(ms, tl)
}

Sporadic(mcp, mrt) /* <<<<<<<< THIS IS NOT CORRECT. */
   if (0.0 + min_processors_required(ms, tl))

/* Returns the # processors required by Cordeiro Dissertation pg 47. 
The .99 is used to approximate a ceiling function. */
INT exported_min_processors_required(permission_set, m,
   id_constraint_set cs) {
   REALToINT(min_processors_required(ms, cs) + 0.99)
}

/*
* id exported Extract_ID_From_IDConstraint(id_constraint idcons) {
   with(idcons) {
      IdConstraintNull: IdNull,
      IdConstraint(op_id, *, *): 
   with(op_id) {
      OperatorIdNull: IdNull,
      OperatorId(*, id, *): id
   }
   }
}
*/

operator_id exported Extract_OpId_From_IDConstraint(id_constraint idcons) {
   with(idcons) {
      IdConstraintNull: OperatorIdNull,
      IdConstraint(op_id, *, *): op_id
   }
}

id_constraint_set exported Id.Constraint.Set.Union(id_constraint_set s1,
   id_constraint_set s2) {
   with(s1) {
      IdConstraintSetNull: s2,
      IdConstraintPair(hd1, tl1):
   with(s2) {
      IdConstraintSetNull: s1,
      IdConstraintPair(hd2, tl2):
      (LessThanOpId(Extract_OpId_From_IDConstraint(hd1),
         Extract_OpId_From_IDConstraint(hd2))
      ? hd1 : Id.Constraint.Set.Union(t1, s2)
      : Id.Constraint.Set.Union(hd2),
      Extract_OpId_From_IDConstraint(hd2))
      ? hd1 : Id.Constraint.Set.Union(t1, tl2)
      : Id.Constraint.Set.Union(s1, tl2)
   }
   }
}

per Get_Per(optional_period o_per) {
   with(o_per) {
      Opt_PeriodNull: PerNull,
      Opt_PeriodPrompt: PerNull,
      Opt_Period(t, *): 
      (Convert_Time2Real(t) >= 0.0)
      ? Per(Convert_Time2Real(t)) : PerNull
   }
}

fw Get_FW(optional_finish_within o_fw) {
   with(o_fw) {
      Opt_FinishWithinNull: FwNull,
      Opt_FinishWithinPrompt: FwNull,
      Opt_FinishWithin(t, *): 
      (Convert_Time2Real(t) >= 0.0)
      ? Fw(Convert_Time2Real(t)) : FwNull
   }
}

mcp Get_Mcp(optional_mcp o_mcp) {
   with(o_mcp) {
      Opt_McpNull: McpNull,
      Opt_McpPrompt: McpNull,
      Opt_Mcp(t, *): 
      (Convert_Time2Real(t) >= 0.0)
      ? Mcp(Convert_Time2Real(t)) : McpNull
   }
}

mrt Get_Mrt(optional_mrt o_mrt) {
   with(o_mrt) {
      Opt_MrtNull: MrtNull,
      Opt_MrtPrompt: MrtNull,
      Opt_Mrt(t, *): 
      (Convert_Time2Real(t) >= 0.0)
      ? Mrt(Convert_Time2Real(t)) : MrtNull
   }
}

met Get_Met(operator_id op_id, op_id_met_set met_set) {
   with(met_set) {
      OpIdMetSetNull: MetNull,
      OpIdMetPair(hd, t1):
   with(hd) {
      OpIdMetNull: Get_Met(op_id, t1),
      OpIdMet(id, the_met):
      (id == op_id)
      ? the_met : Get_Met(op_id, t1)
   }
}

/* The default (assumed) unit associated w/ this int will be microseconds. */
REAL Convert_Time2Real(time t) {
   with(t) {
      TimeNull: -1.0,
      Time(tv, tu): 
      with(tv) {
         IntegerNull: -1.0,
```
APPENDIX D - Auxiliary Functions

```c
IntegerVal(str_digits),
    with(tu) {
        UnitNil: -1.0,
        UnitMICROSECONDS: INTToREAL(STRtoINT(str_digits)),
        UnitNS: INTToREAL(STRtoINT(str_digits)) * 1000.0,
        UnitSEC: INTToREAL(STRtoINT(str_digits)) * 1000000.0,
        UnitMIN: INTToREAL(STRtoINT(str_digits)) * 60000000.0,
        UnitHOURS: INTToREAL(STRtoINT(str_digits)) * 3600000000.0
    }
}

fw Get_FW_from_Per(optional_period o_per) {
    with(o_per) {
        OptPeriodNull: FwNull,
        OptPeriodPrompt: FwNull,
        OptPeriod(t, *),
        ExecutionTimeSEL(t) >= 0.0
    ? Fw(ExecutionTimeSEL(t)) : FwNull
    }
}

per Get_Per_from_FW(optional_finish_within o_fw) {
    with(o_fw) {
        OptFinishWithinNull: PerNull,
        OptFinishWithinPrompt: PerNull,
        OptFinishWithin(t, *),
        ExecutionTimeSEL(t) >= 0.0
    ? Per(ExecutionTimeSEL(t)) : PerNull
    }
}

mcp Get_Mcp_from_Mrt(optional_mrt o_mrt, operator_id op_id, op_id met_set met_set) {
    with(o_mrt) {
        OptMrtNull: McpNull,
        OptMrtPrompt: McpNull,
        OptMrt(mrt_time, *),
        with(met_set) {
            OptMrtSetNull: McpNull,
            OptMrtPair(hd, tl),
            with(hd) {
                OptIdMrtNull: Get_Mrt_from_Mcp(o_mcp, op_id, met_set, hd),
                OptIdMrt(old, met),
                old == op_id
            ? with(met) {
                MetNull: McpNull,
                MET(met_num)
            ? (Convert_Time2Real(mrt_time) >= 0.0)
            ? Mcp(Convert_Time2Real(mrt_time) - met_num) : McpNull
            }
            ? Get_Mcp_from_Mrt(o_mrt, op_id, tl)
        }
    }
}

id_constraint_set exported Get_Id_Constraint_Set(a_constraint ac, op_id met_set met_set) {
    with(ac) {
        IdConstraintNull: IdConstraintSetNull,
        AConstraint(op_id, a_trigger, o_per, o_fw, o_mcp, o_mrt, *),
        (Valid_T_Constraint(ac))
    ? (Is_Per(o_per))
    ? (Is_FW(o_fw))
    ? IdConstraintPair
```
APPENDIX D - Auxiliary Functions

```c
OperatorIdNull; IdNull,
OperatorId(*, id, *): id ) };

operator_id exported Extract_OpId_From_OpIdNet(op_id_met idmet) { with(idmet) { OIdNetOfMonth; OperatorIdNull,
OIdNet(op_id, constraints): op_id } };

cpp_id_met_set exported Op_Id_Met_Set_Union(op_id_met_set s1, op_id_met_set s2) { with(s1) { OIdNetSetNull: s2,
OIdNetPair(hd1, t1l): with(s2) { OIdNetSetNull: s1,
OIdNetPair(hd2, t2l): (LessThanOpId(Extract_OpId_From_OpIdNet(hd1),
Extract_OpId_From_OpIdNet(hd2))) } } };

cpp_id_met_set exported Get_Id_Met_Set(a_vertex vertex) { with(vertex) { AVertexNull; OIdNetSetNull,
AVertex(op_id, opt_time): with(op_time) { OptionalTimeNull: OIdNetSetNull,
OptionalTimePrompt: OIdNetSetNull,
OptionalTime(t): (Convert_Time2Real(t) >= 0.0)
OIdNetPair(OIdNet(op_id, MET{Convert_Time2Real(t)}),
OIdNetSetNull) } } };

id_constraint_set exported Build_Constrained_op_SET(op_id_met_set OpMetSet,
id_constraint_set IdConsSet) { };

edge_set exported Edge_Set_Union(edge_set s1, edge_set s2) { with(s1) { EdgeSetNull: s2,
EdgePair(hd1, t1l): } };

id exported Extract_Id_From_OpIdNet(op_id_met idmet) { with(idmet) { OIdNetNull: IdNull,
OIdNet(op_id, constraints): with(op_id) { } } };

};

id_constraint_set exported Get_Id_Constraint_Set(a_constraint ac, op_id_met_set met_set) { with(ac) { AConstraintNull: IdConstraintSetNull,
AConstraint(op_id, *, o_per, o_hw, o_mcp, o_mrt, *, *, *): IdConstraintSetNil } };

*************** end test *******************/

/* - - - - - - - */

id exported Extract_Id_From_OpIdNet(op_id_met idmet) { with(idmet) { OIdNetNull: IdNull,
OIdNet(op_id, constraints): with(op_id) { } } };

};

id exported Extract_Id_From_OpIdNet(op_id_met idmet) { with(idmet) { OIdNetNull: IdNull,
OIdNet(op_id, constraints): with(op_id) { } } };

239
with(s2) {
  EdgeSetNil; s1,
  EdgePair(hd2, t12);
  (LessThanOpId(Extract_OpId_From_Edge(hd1),
    Extract_OpId_From_Edge(hd2))
  ? hd1 :: Edge_Set_Union(t11, s2)
  : (EqualOpId(Extract_OpId_From_Edge(hd1),
     Extract_OpId_From_Edge(hd2))
  ? hd1 :: Edge_Set_Union(t11, t12)
  : hd2 :: Edge_Set_Union(s1, t12)
  )
  )
}

operator_id Extract_OpId_From_Edge(edge e) {
  with(e) {
    EdgeNull: OperatorIdNull,
    Edge(edge_id, *, *, *): OperatorId(OptionalTypeIdNull,
                                         edge_id,
                                         OperatorIdPairsNull)
  }
}

edge_set exported Get_Edge_Set(an_edge e) {
  with(e) {
    AnEdgeNull: EdgeSetNil,
    AnEdge(id, lt, fvi, tvi): with(fvi) {
      FVertexIdNull: EdgeSetNil,
      FVertexId(foti, fdi, foip): with(tvi) {
        TVertexIdNull: EdgeSetNil,
        TVertexId(toti, tid, toip): with(lt) {
          LatencyTimeNull: EdgePair(EdgeId,
                                   LatencyNull,
                                   Producer(OperatorId(foti, fdi, foip)),
                                   Consumer(OperatorId(toti, tid, toip)),
                                   EdgeSetNil),
          LatencyTimePrompt: EdgePair(EdgeId,
                                   LatencyNull,
                                   Producer(OperatorId(foti, fdi, foip)),
                                   Consumer(OperatorId(toti, tid, toip)),
                                   EdgeSetNil),
          LatencyTime(t): EdgePair(EdgeId,
                                   Latency(Convert_Time2Real(t)),
                                   Producer(OperatorId(foti, fdi, foip)),
                                   Consumer(OperatorId(toti, tid, toip)),
                                   EdgeSetNil),
          }
        }
      }
    }
  }
}
APPENDIX E - Unparsing Rules

/* declare an output_view to output a clean MDL program */
view SHOW_GRAPH_TEXT_VIEW, SDE_VIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW;

prototype
  Prot[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW * : : error_header
multiple_root_message multiple_root_ids
multiple_op_spec_message multiple_op_spec
multiple_type_spec_message multiple_type_spec
also_op_type_message also_op_type_ids
undefined_op_spec_message undefined_op_spec_set
undefined_type_op_spec_message undefined_type_op_spec_set
multiple_vertices_message multiple_vertices
multiple_streams_message multiple_streams
unprocessor_unschedulability_msg
min_processor_msg

  [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW * : : 0]
  ;

pml_components
  ; Pad1[1][SDE_VIEW, SHOW_GRAPH_TEXT_VIEW * : :]
  ; PadPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
  ; IMPLONLY_VIEW * : : ['tn' 10]]
  ;

component
  ; NoComponent[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW * : :<COMPONENT LIST>] *
  ; Op[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW * : :<OPERATOR * 0]
    error_header
    root_message
    multiple_root_message
    multiple_op_spec_message
    also_defined_as_type_message
    undefined_input_message undefined_input_id
    obsolete_input_message obsolete_input_id
    undefined_output_message undefined_output_id
    obsolete_output_message obsolete_output_id
    input_type_error_message input_type_error_set
    output_type_error_message output_type_error_set
    obsolete_state_message obsolete_state_id
    meta_error_message
    obsolete_stream_message obsolete_stream
    undefined_constraint_message undefined_constraint
    obsolete_constraint_message obsolete_constraint
    error_trailer 0 0]
  ; [BASEVIEW * : :<OPERATOR * 0 0]
  ; [SPEC_ONLY_VIEW * : :<OPERATOR * 0 0]
  ; [IMPL_ONLY_VIEW * : :<tn * 0 0]
  ;

  ; Data[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW * : :<TYPE * 0]
    error_header
    multiple_type_spec_message
    also_defined_as_type_message
    undefined_op Implicit message undefined_op_impl
    obsolete_op Impl_message obsolete_op_impl
    error_trailer 0 0]
  ; [BASEVIEW * : :<tn * 0 0]
  ; [SPEC_ONLY_VIEW * : :<tn * 0 0]
  ; [IMPL_ONLY_VIEW * : :<tn * 0 0]
  ;

id: IDNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW * : :<identifier> *]
  [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW * : :<UNDEFINED_ID>]
  ; ID[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
  ; IMPLONLY_VIEW * : :<""]
  ;

integer: IntegerNull [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
  IMPLONLY_VIEW * : :<"]
  ; IntegerVal[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
  IMPLONLY_VIEW * : :<"]
  ;

type_spec
  ; TypeSpec[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
  IMPLONLY_VIEW * : :<"typename" * 0]
    /Generic params*/
    /0_type_decl/ * 0
    /0_operators*/
    /0_keywords*/
    /0_informal_descs*/
    /0_formal_descs*/
    /tnEND*/
  ;

generic_params
  ; GenericName[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW * : :]
    ; GenericPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW * : :<optional generic
parameters> * 0]
    ; Generic[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
    IMPLONLY_VIEW * : :<""* generic*]
    /tn" * 0
    /tnEND*/
  ;
type_declarations
    ; TypeDecl1[1][SDE_VIEW, SHOW_GRAPH_TEXT_VIEW * : :]
    ; TypeDeclPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
    IMPLONLY_VIEW * : :<""]
  ;
a_decl: ADecl[1][SDE_VIEW, SHOW_GRAPH_TEXT_VIEW * : :<type declaration> *]
  ; ADecl[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
  IMPLONLY_VIEW * : :<""]
    /0 * 0
    /0ὦ /id_list*/
    /0"* undefined ADT decl type_name*/
  ;
type_name
  ; TypeNameNull [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW * : :<"type name"]
    [BASEVIEW, SPEC_ONLY_VIEW, IMPLONLY VIEW * : :<UNDEFINED_TYPE_NAME>]
    ; TypeName[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY VIEW,
    IMPLONLY VIEW * : :<""
    /0"* bracket_type_declarations*/
  ;
decl_type_name
  ; DTypeNameNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW * : :<decl type name>] *
  ; DName[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY VIEW,
  IMPLONLY VIEW * : :<""
  ; DTypeInteger[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY VIEW,
  IMPLONLY VIEW * : :<"integer"]
  ;

241


APPENDIX E - Unparsing Rules

| DTypeReal | SView, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY VIEW 0::="REAL" |
| DTypeBoolen | SView, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY VIEW 0::="BOOLEAN" |
| DTypeSimpleId | SView, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY VIEW 0::="" |
| DTypeUserDefined | SView, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY VIEW 0::="" |

@*/id*/
@*/bracket_type_declarations*/

@*/ob_type*/
| OBTypeNone | SView, SHOW_GRAPH_TEXT_VIEW 0::="" |
| OBTypePrompt | SView, SHOW_GRAPH_TEXT_VIEW 0::="tn[type declarations]" |
| OBTypeDeclaration | SView, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY VIEW, IMPL_ONLY VIEW 0::="tn|tn* |
@*/type_declarations*/
@*/tn* |

@*/bracket_type_declarations*/
| BTypeNull | SView, SHOW_GRAPH_TEXT_VIEW 0::="[<type declarations>]*" |
| BTypeDeclaration | SView, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY VIEW, IMPL_ONLY VIEW 0::="tn|tn* |
@*/type_declarations*/
@*/tn* |

@*/alone_id_list*/
| AIdNil | SView, SHOW_GRAPH_TEXT_VIEW 0::="" |
| AIdPair | SView, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY VIEW, IMPL_ONLY VIEW 0::="[*","*]" |

@*/id_list*/
| IdNil | SView, SHOW_GRAPH_TEXT_VIEW 0::="" |
| IdPair | SView, SHOW_GRAPH_TEXT_VIEW 0::="" | multiply_defined ",tn* @ ] |
| [BASEVIEW, SPEC_ONLY VIEW, IMPL_ONLY VIEW 0::="[*",tn* @ ] |

@*/o_type_decls*/
| TypeNone | SView, SHOW_GRAPH_TEXT_VIEW 0::="" |
| TypePrompt | SView, SHOW_GRAPH_TEXT_VIEW 0::="" | [optional type declarations]|tb* |
| Type | SView, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY VIEW 0::="tn|tn* |
@*/tb* |
@*/type_declarations*/

@*/o_operators*/
| OperatorNil | SView, SHOW_GRAPH_TEXT_VIEW 0::="" |
| OperatorPair | SView, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY VIEW, IMPL_ONLY VIEW 0::="[*"] |

@*/top_spec*/
| TopSpecNil | SView, SHOW_GRAPH_TEXT_VIEW 0::="" | multiply_defined_trailer |
| TopSpec | SView, SHOW_GRAPH_TEXT_VIEW 0::="" | operator_spec |
@*/id*/
@*/top_spec*/

@*/operator_spec*/
| OperatorSpec | SView, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY VIEW, IMPL_ONLY VIEW 0::="tn|SPECIFICATION |
| /tn|GENERIC |
| /tn|SPECIFICATION |
| /tn|SPECIFICATION |
| /tn|SPECIFICATION |
@*/o_operators*/

@*/o_inputs*/
| OInputNil | SView, SHOW_GRAPH_TEXT_VIEW 0::="" |
| OInputPair | SView, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY VIEW, IMPL_ONLY VIEW 0::="[*"] |

@*/o_outputs*/
| OOutputNil | SView, SHOW_GRAPH_TEXT_VIEW 0::="" |
| OOutputPair | SView, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY VIEW, IMPL_ONLY VIEW 0::="[*"] |

@*/error_header*/

242
APPENDIX E - Unparsing Rules

/* exceptions_list */

- o_exceptions_list
  - ONullException(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW 0::=)
  - ONullException(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
    IMPL_ONLY_VIEW 0::= "*" 0)

- o_exceptoin
  - OpExceptionNone(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW 0::="*" 0)
  - OpException(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
    IMPL_ONLY_VIEW 0::="*" 0)

/* timing_info */

- O.timing_info
  - O.timingInfoNone(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW 0::=)
  - O.timingInfoPrompt(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW 0::="*" 0)
  - O.timingInfo(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
    IMPL_ONLY_VIEW 0::="*" 0)
    @ "*" 0 /* time */
    @ "*" 0 /* time */
    @ "*" 0 /* time */

/* time */

- TimeNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW 0::="*" 0)
- Time(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
  IMPL_ONLY_VIEW 0::="*" 0)
  @ "*" 0 /* time */

/* time unit */

- UnitNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- UnitMicrosecond(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- UnitHMS(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- UnitSec(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL_ONLY_VIEW 0::="*" 0)
- UnitMin(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- UnitHour(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)

/* o_states_list */

- O_states_listNone(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW 0::=)
- O_states_listPair(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC_ONLY_VIEW,
  IMPL ONLY VIEW 0::="*" 0)

/* o_states */

- O_statesNone(SDE_VIEW, SHOW_GRAPH_TEXT VIEW 0::="*" 0)
- O_states(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
  @ "*" 0 /* states */
  @ "*" 0 /* expression_list */

/* initial_arg */

- InitialArgNone(SDE_VIEW, SHOW_GRAPH_TEXT VIEW 0::=)
- InitialArg(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)

/* an_argument */

- AnArgNone(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- AnArg(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)

/* expression_list */

- InitialExpListNone(SDE_VIEW, SHOW_GRAPH_TEXT VIEW 0::=)
- InitialExpList(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)

/* expression */

- ExpNull(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- Identifier(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- Textual_Description(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- TypeExpression(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- ParenthesizedExp(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)

/* BOOLEAN EXPRESSIONS */

- True(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- False(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- NotExp(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- EqualExp(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- LessExp(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- GreaterExp(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- GreaterEqualExp(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- LessEqualExp(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- NotEqualExp(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- AndExp(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- OrExp(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)
- XorExp(SDE_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC ONLY VIEW,
  IMPL ONLY VIEW 0::="*" 0)

/* ARITHMETIC EXPRESSIONS */

243
APPENDIX E - Unparsing Rules

<table>
<thead>
<tr>
<th>Text: TextNull</th>
<th>Text: TextNull</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text: TextNull</td>
<td>Text: TextNull</td>
</tr>
</tbody>
</table>

```
type_impl {
  TypeImplNull {
  
  
  }
  AdatypeImpl {
    
    
  }
  Template {
    
    
  }
  OpImplNull {
    
    
  }
  OperatorImpl {
    
    
  }
}
```

```
/* STRING EXPRESSIONS */

<table>
<thead>
<tr>
<th>ConcatExp</th>
<th>ConcatExp</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConcatExp</td>
<td>ConcatExp</td>
</tr>
</tbody>
</table>

```

```
/* KEYWORDS */

<table>
<thead>
<tr>
<th>KeywordsNone</th>
<th>KeywordsNone</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeywordsNone</td>
<td>KeywordsNone</td>
</tr>
</tbody>
</table>

```

```
/* INFORMAL DESCRIPTION */

<table>
<thead>
<tr>
<th>InformalDesNone</th>
<th>InformalDesNone</th>
</tr>
</thead>
<tbody>
<tr>
<td>InformalDesNone</td>
<td>InformalDesNone</td>
</tr>
</tbody>
</table>

```

```
/* FORMAL DESCRIPTION */

<table>
<thead>
<tr>
<th>FormalDesNone</th>
<th>FormalDesNone</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormalDesNone</td>
<td>FormalDesNone</td>
</tr>
</tbody>
</table>

```

```
/* REQUIREMENTS TRACE */

<table>
<thead>
<tr>
<th>RegstrTraceNone</th>
<th>RegstrTraceNone</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegstrTraceNone</td>
<td>RegstrTraceNone</td>
</tr>
</tbody>
</table>
```

244
APPENDIX E - Unparsing Rules

error_trailer
0 "tb"/"operator_impl"/

[BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ^::="tt"nOPERATOR *
0 /"id"/
0 "tb"*/"operator_impl"/

] graph: GraphNull[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"n<graph><tb>]
| Graph[BASEVIEW ^::="tt"nSHOW_GRAPH_VIEW: see graph viewer for details --tb" 
/vertex_list/
"tn" .. /edge_list/
"nb"
] [SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, 
IMPL_ONLY_VIEW ^::="tt"nGRAPH *
0 /vertex_list/
"tn" @ /edge_list/
"nb"
]

]vertex_list
:VertexListNull[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"
| VertexListPair[BASEVIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, 
IMPL_ONLY_VIEW ^::="tt"n["tn"] 0 ]

] a_vertex
:AVertexNull[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"n(optional vertex list)"
| AVertex[BASEVIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, 
IMPL_ONLY_VIEW ^::="tt"nVERTEX *
0 /operator_id/
0 /optional_time/
"tb"
]

] operator_id
:OperatorIdNull[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"n<operator id>]
| OperatorId[BASEVIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, 
IMPL_ONLY_VIEW ^::="tt"nOPERATOR *
0 /optional_type_id/
0 /"id"/
0]*/operator_id_pairs/

] optional_type_id
:OptionalTypeIdNull[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"
| OptionalTypeIdPrompt[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"n[optional type id] ."]
| OptionalTypeId[BASEVIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, 
IMPL_ONLY_VIEW ^::="tt"n["tn"/*"id"]

] operator_id_pairs
:OperatorIdPairsNull[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"
| OperatorIdPairsPrompt[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"n[optional vertex * 
0]*/operator_id_pairs

] declarations
:Declarations[BASEVIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, 
IMPL_ONLYVIEW ^::="tt"nDATA STREAM *
0]*/optional_timers/

] optional_streams
:OptionalStreamNull[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"n<optional streams>]
| OptionalStreamPrompt[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"n[optional streams]"tb"
| Streams[BASEVIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, 
IMPL_ONLYVIEW ^::="tt"nDATA STREAM

] latency_time
:LatencyTimeNull[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"
| LatencyTimePrompt[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"n[optional latency time]"
| LatencyTime[BASEVIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, 
IMPL_ONLYVIEW ^::="tt"n/*"id"
0 /"latency_time"/
"tb"
| 0]*/vertex_id /
0]*/vertex_id

] from_vertex_id
:FVertexIdNull[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"n<vertex id>]
| FVertexId[BASEVIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, 
IMPL_ONLYVIEW ^::="tt"nOPERATOR *
0 /"optional_type_name"/
0 /"id"/
0]*/operator_id_pairs/

] to_vertex_id
:TVertexIdNull[BASEVIEW, SHOW_GRAPH_TEXT_VIEW ^::="tt"n<vertex id>]
| TVertexId[BASEVIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, 
IMPL_ONLYVIEW ^::="tt"nOPERATOR *
0 /"optional_type_name"/
0 /"id"/
0]*/operator_id_pairs/

]
APPENDIX E - Unparsing Rules

```
"line" @/type_declarations/  
"line"

optional_timers  
| TriggerNull[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:=]  
| TimersPrompt[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:="\n\n[optional timers]\n"]  
| Timers[SDN_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC_ONLY VIEW,  
  IMPL_ONLY VIEW @:="\n\n[optional timers]\n"]  
| "line"  
| "line"  
| "line"

cc: CcNull[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:="\n\n[controls constraints]\n"]  
| CC[SDN_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC_ONLY VIEW,  
  IMPL_ONLY VIEW @:="\n\n[controls constraints]\n"]  
| "line"  
| "line"  
| "line"

constraints  
| ConstraintsNull[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:=]  
| ConstraintsPair[SDN_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC_ONLY VIEW,  
  IMPL_ONLY VIEW @:="\n\n[controls constraints]\n"]  
| "line"  
| "line"  
| "line"

a_constraint  
| AConstraintNull[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:="\n\n[constraint]\n"]  
| AConstraint[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:="\n\n[constraint]\n"]  
| "line"  
| "line"  
| "line"

error_header  
| error_header[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:="\n\n[error_header]\n"]  
| "line"  
| "line"  
| "line"

error_trailer  
| error_trailer[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:="\n\n[error_trailer]\n"]  
| "line"  
| "line"  
| "line"

required_period  
| RequiredPeriod[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:="\n\n[required period]\n"]  
| "line"  
| "line"  
| "line"

optional_period  
| OptionalRequiredPeriod[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:="\n\n[optional period]\n"]  
| "line"  
| "line"  
| "line"

optional_finish_within  
| OptionalFinishWithin[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:="\n\n[optional finish within]\n"]  
| "line"  
| "line"  
| "line"

optional_mcp  
| OptionalMcp[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:="\n\n[optional mcp]\n"]  
| "line"  
| "line"  
| "line"

optional_triggers  
| OptionalTriggersAllOrSome[SDN_VIEW, SHOW_GRAPH_TEXT VIEW, BASEVIEW, SPEC_ONLY VIEW,  
  IMPL_ONLY VIEW @:="\n\n[optional triggers]\n"]  
| "line"  
| "line"  
| "line"

triggers  
| TriggerNull[SDN_VIEW, SHOW_GRAPH_TEXT VIEW @:="\n\n[triggers]\n"]  
| "line"  
| "line"  
| "line"

```

246
APPENDIX E - Unparsing Rules

output guards:
  | OutputGuardNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ::=)
  | OutputGuardPair(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "[*tn*]"

a_guard:
  | AGuard(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ::= "tttn(output guard)tb"
  | AGuard(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "tttnoutput"
  | "tttn\*id_list/"
  | "tttnf* /c_expression/"
  | 2 "bbb/"/regmts_trace/"

exception ops:
  | ExceptionOpsNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ::=)
  | ExceptionOpsPrompt(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ::= "tttn(optional exceptions)tb"
  | Exception(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "[*tn*]"

exception_options:
  | ExceptionOptionsNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ::=)
  | ExceptionOptionsPair(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "[*tn*]"

an_exception:
  | AnExceptionNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ::= "tttnan exception>tb"
  | AnException(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "tttnEXCEPTION"
  | \"*"/id*/
  | \"tttn\*optional_if_predicate/"
  | 0 "bbb/"/regmts_trace/"

timer_operations:
  | TimerOperationsNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ::=)
  | TimerOperationsPair(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "[*tn*]"

a_timer_operation:
  | ATimerOperationNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ::= "tttn(timer operation)tb"
  | ATimerReset(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "tttnRESET"
  | \"*"/id*/
  | \"tttn\*optional_if_predicate/"
  | 0 "bbb/"/regmts_trace/"

  | ATimerStart(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "tttnSTART"
  | \"*"/id*/
  | \"tttn\*optional_if_predicate/"
  | 0 "bbb/"/regmts_trace/"

  | ATimerStop(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "tttnSTOP"
  | \"*"/id*/
  | \"tttn\*optional_if_predicate/"
  | 0 "bbb/"/regmts_trace/"

optional_if_predicate:
  | OptIfPredicateNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ::=)
  | OptIfPredicatePrompt(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ::= "tttn(optional if predicate)"

  | OptIfPredicate(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "tttnf*"
  | \"/c_expression/"

  /* Conditional for IP expressions */

  c_initial_arg:
  | CIntInitialArgNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ::=)
  | CIntInitialArg(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "[*tn*]"

  c_an_argument:
  | CanArgNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "[*tn*]"
  | CanArg(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "tttn 0 "bbb"

  c_expression_list:
  | CIntInitialExpListNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ::=)
  | CIntInitialExpListPair(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "[*tn*]"

  c_expression:
  | CExpNull(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "<exp>
  | CIdentifier(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "<exp>"
  | CTextual_Description(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "<exp>"
  | CTupleExpression(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "<exp>"
  | CTimeExpression(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "<exp>"
  | CParenthesizedExpression(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "<exp>"

  /* BOOLEAN EXPRESSIONS */

  | CTrue(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "TRUE"
  | CFalse(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "FALSE"
  | CNotExp(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "NOT "
  | CEqExp(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "<exp>"
  | CLessExp(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "<exp>
  | CGreaterExp(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "<exp>
  | CGreatEqualExp(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "<exp>
  | CLessEqualExp(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "<exp>
  | CNotEqualExp(SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ::= "<exp>

247
APPENDIX E - Unparsing Rules

/
CIntExp[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * / 0]
CReal[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * / 0]
CPlusExp[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * + 0]
CMinusExp[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * - 0]
CtimesExp[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * * 0]
CDivExp[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * / 0]
CNegativeExp[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * - 0]
CPositiveExp[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * + 0]
CAbsExp[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * abs(*)]
CRemExp[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * rem(*)]
CModExp[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * mod(*)]
CExponentExp[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * ** 0]

/* STRING EXPRESSIONS */
CConcatExp[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY View 0::0 * & 0]

/* unformatted text */

commentLines:
CommentLinesNil[SD_E, SHOW_GRAPH_TEXT_VIEW 0]
CommentLinesPair[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::0 * "\n 0]

commentLine:
CommentLineNil[SD_E, SHOW_GRAPH_TEXT_VIEW 0::*unformatted text*]
CommentLine[SD_E, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW 0::*unformatted text*]

optionalComment:
OptionalCommentNil[SD_E, SHOW_GRAPH_TEXT_VIEW 0]
OptionalCommentPrompt[SD_E, SHOW_GRAPH_TEXT_VIEW 0::*unformatted text*]
OptionalComment[SDL_E, SHOW_GRAPH_TEXT_VIEW 0::*unformatted text*]
APPENDIX F - Transformation Rules

transform component
  on "Type"<component>::Data{
    <id>,
    <type_spec>,
    <type_impl>
  },
  on "Operator"<component>::Op{
    <id>,
    <operator_spec>,
    <operator_impl>
  }

transform type_spec
  on "TypeSpec"<type_spec>::TypeSpec{
    <co_generic_params>,
    <co_type_decls>,
    <co_operators>,
    <co_keywords>,
    <co_informal_descs>,
    <co_formal_descs>
  }

transform co_generic_params
  on "Generic"<co_generic_params>::Generic{
    <type_declarations>
  }

transform co_type_decls
  on "Type"<co_type_decls>::Type{
    <type_declarations>
  }

transform t_oper_spec
  on "Operators"<t_oper_spec>::OpeSpec{
    <id>,
    <operator_spec>
  }

transform time_unit
  on "MICROSECOND"<time_unit>::UnitMICROSECONDS(),
  on "NS"<time_unit>::UnitNS(),
  on "SEC"<time_unit>::UnitSEC(),
  on "MINUTE"<time_unit>::UnitMIN(),
  on "HOURS"<time_unit>::UnitHOURS()

transform co_keywords
  on "Keywords"<co_keywords>::KeyWords{
    <alone_id_list>
  }

transform co_informal_descs
  on "Informal_Descriptions"<co_informal_descs>::InformalDescs{
    <comment_lines>
  }

transform o_formal_descs
  on "Formal_Descriptions"<o_formal_descs>::FormalDescs{
    <comment_lines>
  }

transform operator_spec
  on "Operator"<operator_spec>::OperatorSpec{
    <co_generic_list>,
    <co_inputs_list>,
    <co_outputs_list>,
    <co_states_list>,
    <co_exceptions_list>,
    <co_timing_info>,
    <co_keywords>,
    <co_informal_descs>,
    <co_formal_descs>
  }

transform co_generics
  on "OptionalGenerics"<co_generics>::OpGenerics{
    <type_declarations>,
    <regtms_trace>
  }

transform co_inputs
  on "OptionalInputs"<co_inputs>::OpInputs{
    <type_declarations>,
    <regtms_trace>
  }

transform co_outputs
  on "OptionalOutputs"<co_outputs>::OpOutputs{
    <type_declarations>,
    <regtms_trace>
  }

transform co_states
  on "OptionalStates"<co_states>::OpStates{
    <type_declarations>,
    <expression_list>,
    <regtms_trace>
  }

transform an_argument
  on "Exp_Argument"<an_argument>::AnArgument{
    <expression_list>
  }

249
APPENDIX F - Transformation Rules

```plaintext
transform o_exceptions
  on "Optional_Exceptions"
    <o_exceptions>:
      OpExceptions{
        <alone_id_list>,
        <regmts_trace>
      }
    
transform o_timing_info
  on "Optional_Timing_Info"
    <o_timing_info>:
      OptimingInfo{
        <time>,
        <regmts_trace>
      }
    
transform time
  on "Time_Expression"
    <time>:
      Time{
        <integer>,
        <time_unit>
      }
    
transform regmts_trace
  on "RegmtsTrace"
    <regmts_trace>:
      ReqntsTrace{
        <alone_id_list>
      }
    
/*
 ------- TYPE DECLARATION TRANSFORMATIONS
 -------
 -------*/

transform a_decl
  on "DeclList"<a_decl>:ADecl{
    <id_list>,
    <decl_type_name>
  }

transform type_name
  on "TypeName"<type_name>: TypeName{
    <id>,
    <co_bracket_type_declarations>
  }

transform decl_type_name
  on "INTEGER"<decl_type_name>: DTypeInteger(),
  on "FLOAT"<decl_type_name>: DTypeSimpleId(Id("FLOAT")),
  on "BOOLEAN"<decl_type_name>: DTypeBoolean(),
  /*
  on "EXCEPTION"<decl_type_name>: DTypeException(),
  */
  on "REAL"<decl_type_name>: DTypeReal(),
  on "UserDefined"
    <decl_type_name>: DTypeUserDefined{
      <id>,
      <bracket_type_declarations>
    },
  on "REAL"<decl_type_name>: DTypeReal()
}

transform o_bracket_type_declarations
  on "[Type Declarations]"
    <o_bracket_type_declarations>:
      OTypeDeclaration{
        <type_declarations>
      }
    
transform bracket_type_declarations
  on "Type Declarations"
    <bracket_type_declarations>:BTypeDeclaration{
      <type_declarations>
    }

/*
----------- EXPRESSION TEMPLATE TRANSFORMATIONS
*/

transform expression
  on "Ident"<expression>: Identifier<id>,
  on "TypeExpression"
    <expression>: TypeExpression{
      <type_name>,
      <id>,
      <initial_args>
    },
  on "Textual"<expression>: Textual_Description<commentLines>,
  on "Textual"<expression>: Textual_Description<string_lit>,
  on "Parens"<expression>: ParenthesizedExp<expression>,
  on "Boolean"<expression>: BooleanExp<expression>,
  on "And"<expression>: AndExp{
    <expression>,
    <expression>
  },
  on "Or"<expression>: OrExp{
    <expression>,
    <expression>
  },
  on "Xor"<expression>: XorExp{
    <expression>,
    <expression>
  },
  on "True"<expression>: True(),
  on "False"<expression>: False()
*/
```
APPENDIX F - Transformation Rules

on "Not"<expression>: NotExp{
  <expression>
},

on "In"<expression>: EqualExp{
  <expression>,
  <expression>
},

on "Less"<expression>: LessExp{
  <expression>,
  <expression>
},

on "Greater"<expression>: GreaterExp{
  <expression>,
  <expression>
},

on "Non"<expression>: NotEqualExp{
  <expression>,
  <expression>
},

on "Plus"<expression>: PlusExp{
  <expression>,
  <expression>
},

on "Minus"<expression>: MinusExp{
  <expression>,
  <expression>
},

on "Times"<expression>: TimesExp{
  <expression>,
  <expression>
},

on "Div"<expression>: DivExp{
  <expression>,
  <expression>
},

on "Negative"<expression>: NegativeExp{
  <expression>
},

on "Mod"<expression>: ModExp{
  <expression>,
  <expression>
},

on "Rem"<expression>: RemExp{
  <expression>,
  <expression>
},

on "Exponent"<expression>: ExponentExp{
  <expression>,
  <expression>
},

/* string expressions */
on "&"<expression>: ConcatExp{
  <expression>,
  <expression>
}

/* TEMPLATE TRANSFORMATIONS FOR NEW PRODUCTIONS */
transform type_impl
  on "Ada_Implementation" 
    <type_impl>:Get_Ada_Type_Impl(),
    <id>
  ),

/* */
transform t_op_impl
  on "Operator_Impl" 
    <t_op_impl>:TopImpl!
    <id>,
    <operator_impl>
  );

transform operator_impl
  on "Ada_Implementation"
    <operator_impl>:Get_Ada_Op_Impl(),
    <id>
  ),

/* */
transform "PSDL_Implementation"
  <operator_impl>:OperatorImpl{
    <graph>,
    <declarations>,
    <id>
APPENDIX F - Transformation Rules

transform from_vertex_id
  on "Vertex_ID"<from_vertex_id>:FVertexId(
    <optional_type_id>,
    <id>,
    <operator_id_pairs>
  )
;

transform to_vertex_id
  on "Vertex_ID"<to_vertex_id>:TVertexId(
    <optional_type_id>,
    <id>,
    <operator_id_pairs>
  )
;

transform declarations
  on "Declarations"<declarations>:Declarations(
    <optional_streams>,
    <optional_timers>
  )
;

transform optional_streams
  on "Streams"<optional_streams>:Streams(
    <type_declarations>
  )
;

transform optional_timers
  on "Timers"<optional_timers>:Timers(
    <alone_id_list>
  )
;

transform cc
  on "ControlConstraint"<cc>:Cc(
    <constraint>,
    <io_informal_desc>
  )
;

transform a_constraint
  on "A_Constraint"<a_constraint>:AConstraint(
    <operator_id>,
    <optional_trigger>,
    <optional_period>,
    <optional_finish_within>,
    <optional_mcp>,
    <optional_art>,
    <output_guard>,
    <exception_op>,
    <timer_operations>
  )
;

transform optional_trigger
  on "ALL/SOME-Trigger"<optional_trigger>:OptionalTriggerAllOrSome(
    <type_of_trigger>,
    <alone_id_list>
  )
APPENDIX G - Concrete Rules

PROTOTYPE(synthesized prototype t);  
PSDL_COMPONENTS(  
  synthesized pSDL_components reversed;  
);  
COMPONENT(synthesized component t);  
ID (synthesized id t);  
INTEGER(synthesized integer t);  
TYPE_SPEC(synthesized type_spec t);  
O_GENERIC_PARAMS(synthesized o_generic_params t);  

TYPE_DECLARATIONS  
  inherited type_declarations tail;  
  synthesized type_declarations reversed;  
);  
A_DECL(synthesized a_decl t);  
TYPE_NAME(synthesized type_name t);  
DECL_TYPE_NAME(synthesized decl_type_name t);  
BRACKET_TYPE_DECLARATIONS  
  synthesized bracket_type_declarations t);  
O_BRACKET_TYPE_DECLARATIONS  
  synthesized o_bracket_type_declarations t);  
ALONE_ID_LIST(  
  inherited alone_id_list tail;  
  synthesized alone_id_list reversed;  
);  
ID_LIST  
  inherited id_list tail;  
  synthesized id_list reversed;  
);  
O_TYPE_DECLS(synthesized o_type_decls t);  
O_OPERATORS(  
  inherited o_operators tail;  
  synthesized o_operators reversed;  
);  
T_OPER Spec(synthesized t_oper_spec t);  
OPERATOR_SPEC(synthesized operator_spec t);  
O_GENERICS LIST(  
  inherited o_generics_list tail;  
  synthesized o_generics_list reversed;  
);  
O_GENERICS(synthesized o_generics t);  
O_INPUTS LIST(  
  inherited o_inputs_list tail;  
  synthesized o_inputs_list reversed;  
);  
O_INPUTS(synthesized o_inputs t);  
O_OUTPUTS_LIST(  
  inherited o_outputs_list tail;  
  synthesized o_outputs_list reversed;  
);  
O_OUTPUTS(synthesized o_outputs t);  
O_EXCEPTIONS_LIST(  
  inherited o_exceptions_list tail;  
  synthesized o_exceptions_list reversed;  
);  
O_EXCEPTIONS(synthesized o_exceptions t);  
O_TIMING_INFO(synthesized o_timing_info t);  
TIME_UNIT(synthesized time_unit t);  
TIME (synthesized time t);  
O_STATES_LIST(  
  inherited o_states_list tail;  
  synthesized o_states_list reversed;  
);  
O_STATES(synthesized o_states t);  
INITIAL_ARGS(  
  inherited initial_args tail;  
  synthesized initial_args reversed;  
);  
AN_ARGUMENT(synthesized an_argument t);  
C_INITIAL_ARGS(  
  inherited c_initial_args tail;  
  synthesized c_initial_args reversed;  
);  
C_AN_ARGUMENT(synthesized c_an_argument t);  
EXPRESSION_LIST(  
  inherited expression_list tail;  
  synthesized expression_list reversed;  
);  
EXPRESSION(synthesized expression t);  
C_EXPRESSION_LIST(  
  inherited c_expression_list tail;  
  synthesized c_expression_list reversed;  
);  
C_EXPRESSION(synthesized c_expression t);  
O_KEYWORDS(synthesized o_keywords t);  
O_INFORMAL_DESC(synthesized o_informal_descs t);  
O_FORMAL_DESC(synthesized o_formal_descs t);  
RECURS_TRACE(synthesized recurs_trace t);  

"NEW CONCRETE INPUT SYMBOLS"

255
APPENDIX G - Concrete Rules

c_states_list - O_STATES_LIST.reversed
[O_STATES_LIST.tail=StatesListNone];

c_states - O_STATES.t;

an_argument - AN_ARGUMENT.t;
c_an_argument - C_AN_ARGUMENT.t;

initial_arg - INITIALARGS.reversed
(INITIALARGS.tail=InitialArgsNil);

c_initial_arg - C_INITIALARGS.reversed
(C_INITIALARGS.tail=InitialArgsNil);

expression_list - EXPRESSION_LIST.reversed
(EXPRESSION_LIST.tail=InitialExprListNil);

expression - EXPRESSION.t;

c_expression_list - C_EXPRESSION_LIST.reversed
(C_EXPRESSION_LIST.tail=InitialExprListNil);

c_expression - C_EXPRESSION.t;

o_keywords - O_KEYWORDS.t;
o_informal_descs - O_INFORMAL_DESCS.t;
o_formal_descs - O_FORMAL_DESCS.t;
oreqts_trace - REQTS_TRACE.t;
type_impl - TYPE_IMPL.t;
operator_impl - OPERATOR_IMPL.t;

operator_impl_list - OPERATOR_IMPL_LIST.reversed
(OPERATOR_IMPL_LIST.tail=ImplListNil);

goal - GOAL.t;
declarations - DECLARATIONS.t;
cc - CC.t;
t_op_impl - T_OPL_LIST.t;

vertex_list - VERTEX_LIST.reversed
(VERTEX_LIST.tail=VertexListNil);

a_vertex - A_VERTEX.t;
operator_id - OPERATOR_ID.t;
optional_type_id - OPTIONAL_TYPE_ID.t;
operator_id_pairs - OPERATOR_ID_PAIRS.t;
optional_time - OPTIONAL_TIME.t;
latency_time - LATENCY_TIME.t;

deedge_list - DEGDE_LIST.reversed
(DEGDE_LIST.tail=EdgeListNil);

an_edge - AN_EDGE.t;
from_vertex_id - FROM_VERTEX_ID.t;
to_vertex_id - TO_VERTEX_ID.t;
optional_streams - OPTIONAL_STREAMS.t;
optional_timers - OPTIONAL_TIMERS.t;
constraints - CONSTRAINTS.reversed
(CONSTRAINTS.tail=ConstraintNil);

a_constraint - A_CONSTRAINT.t;
optional_trigger - OPTIONAL_TRIGGER.t;
type_of_trigger - TYPE_OF_TRIGGER.t;

optional_period - OPTIONAL_PERIOD.t;
optional_finit within - OPTIONAL_FINISH_WITHIN.t;
optional_mcp - OPTIONAL_MCP.t;
optional_rmt - OPTIONAL_RMT.t;

output guards - OUTPUT_GUARDS.reversed
(OUTPUT_GUARDS.tail=OutputGuardNil);

a_guard - A_GUARD.t;
exception_opc - EXCEPTION_OPCODE.t;

exception_options - EXCEPTION_OPTIONS.reversed
(EXCEPTION_OPTIONS.tail=ExceptionOptionsNil);

timer_operations - TIMER_OPERATIONS.reversed
(TIMER_OPERATIONS.tail=TimerOperationsNil);

a_timer_operation - A_TIMER_OPERATION.t;
optional_if Predicate - OPTIONAL_IF_PREDICATE.t;

/ * ------------------------ PRECEDENCE DECLARATIONS ------------------------ */

left ANDB ORB XORB;
left '<' '>' '=' GETB LTB ltEB NBQEB;
left '+' '-' '+';
left '*' '/' MODB REMB;
left EXPB ABSB NXYB;

/ */

/* CONCRETE GRAMMAR'S PRODUCTIONS */

PROTOTYPE ::= (PSDL COMPONENTS)
{;
    PROTOTYPE.t=Prot(PSDL COMPONENTS.reversed);
}
;

PSDL COMPONENTS ::= ()(PSDL COMPONENTS reversed = PSDLNil)
| (COMPONENT PSDL COMPONENTS)
    PSDL COMPONENTS$1 reversed= COMPONENT.t:PSDL COMPONENTS$2.reversed;
    |
;
COMPONENT ::= (TYPEB ID SPEC TYPE_IMPL)
    COMPONENT.t = (Data(;
        ID.t, TYPE_SPEC.t,
        TYPE_IMPL.t)
    )
|;

257
APPENDIX G - Concrete Rules

A_DECL ::= (ID_LIST ' ; ' DECL_TYPE_NAME)
          (ID_LIST.tail = IDNil)
          A_DECL.t = ADecl(ID_LIST.reversed, DECL_TYPE_NAME.t);
          
          TYPE_NAME ::= (ID O_BRACKET_TYPE_DECLARATIONS)
          (TYPE_NAME.t =TypeName{
          ID.t,
          O_BRACKET_TYPE_DECLARATIONS.t
          });
          
          DECL_TYPE_NAME ::= (INTNOW)
          (DECL_TYPE_NAME.t = DTypeInteger());
          
          (REALNOW)
          (DECL_TYPE_NAME.t = DTypeReal());
          
          (BOOLNOW)
          (DECL_TYPE_NAME.t = DTypeBoolean());
          
          (ID)
          (DECL_TYPE_NAME.t = DTypeSimpleId(ID.t));
          
          (ID_BRACKET_TYPE_DECLARATIONS)
          (DECL_TYPE_NAME.t = DTypeUserDefined{
          ID.t,
          BRACKET_TYPE_DECLARATIONS.t});
          
          BRACKET_TYPE_DECLARATIONS ::= [' ' TYPE_DECLARATIONS ' ']
          (TYPE_DECLARATIONS.tail = TypeDeclNil;
          BRACKET_TYPE_DECLARATIONS.t =
          BTypeDeclaration{
          TYPE_DECLARATIONS.reversed
          });
          
          O_GENERIC_PARAMS ::= ()(O_GENERIC_PARAMS.t = (GenericNone));
          
          (GENERICKW TYPE_DECLARATIONS)
          (TYPE_DECLARATIONS.tail = TypeDeclNil;
          O_GENERIC_PARAMS.t = Generic{
          TYPE_DECLARATIONS.reversed
          });
          
          TYPE_DECLARATIONS ::= (A_DECL)
          (TYPE_DECLARATIONS.reversed =
          A_DECL.t::TYPE_DECLARATIONS.tail
          );
          
          (TYPE_DECLARATIONS ' ; ' A_DECL)
          (TYPE_DECLARATIONS$2.tail =
          A_DECL.t::TYPE_DECLARATIONS$1.tail
          );
          TYPE_DECLARATIONS$1.reversed=
          TYPE_DECLARATIONS$2.reversed;
          
          ALONE_ID_LIST ::= ()(ALONE_ID_LIST$1.reversed = AIdNil);
          
          (ID)
          (ALONE_ID_LIST$1.reversed = (ID.t::ALONE_ID_LIST$1.tail));
          
          (ALONE_ID_LIST ' ; ' ID)
          (ALONE_ID_LIST$2.tail = (ID.t::ALONE_ID_LIST$1.tail);
          ALONE_ID_LIST$1.reversed=ALONE_ID_LIST$2.reversed;
APPENDIX G - Concrete Rules

ID_LIST ::= (ID)
           (ID_LIST$1.reversed(ID.t::ID_LIST$1.tail));

   | (ID_LIST ',' ID)
   |   ID_LIST$2.tail=(ID.t::ID_LIST$2.tail);
   | ID_LIST$1.reversed=ID_LIST$2.reversed;
   |
O_TYPE_DECLS
 ::= ()(O_TYPE_DECLS.t = (TypeNone));
   | (TYPE_DECLARATIONS)
   |   (TYPE_DECLARATIONS.tail=TypeDeclNil);
   | O_TYPE_DECLS.t = Type(TYPE_DECLARATIONS.reversed);
   |
O_OPERATORS
 ::= ()(O_OPERATORS.reversed=(O_OPERATORS.tail));
   | (O_OPERATORS T_OPER_SPEC)
   |   O_OPERATORS$2.tail={
   |       T_OPER_SPEC.t = TOpSpec{
   |           ID.t,
   |           OPERATOR_SPEC.t
   |       };
   |
T_OPER_SPEC
 ::= (OPERATOR$1 ID OPERATOR_SPEC)
   | (T_OPER_SPEC.t = TOpSpec{
   |   ID.t,
   |   OPERATOR_SPEC.t
   | });
|
OPERATOR_SPEC
 ::= (SPEC)
   | O_GENERICS_LIST
   | O_INPUTS_LIST
   | O_OUTPUTS_LIST
   | O_STATES_LIST
   | O_EXCEPTIONS_LIST
   | O_TIMING_INFO
   | O_KEYWORDS
   | O_INFORMAL_DESCS
   | O_FORMAL_DESCS
END()
   | (O_GENERICS_LIST.tail=GenericsListNone;
   | O_INPUTS_LIST.tail=InputsListNone;
   | O_OUTPUTS_LIST.tail=OutputsListNone;
   | O_STATES_LIST.tail=StatesListNone;
   | O_EXCEPTIONS_LIST.tail=ExceptionListNone;
   | OPERATOR_SPEC.t = OperatorSpec{
   |   O_GENERICS_LIST.reversed,
   |   O_INPUTS_LIST.reversed,
   |   O_OUTPUTS_LIST.reversed,
   |   O_STATES_LIST.reversed,
   |   O_EXCEPTIONS_LIST.reversed,
   |   O_TIMING_INFO.t,
   |   O_KEYWORDS.t,
   |   O_INFORMAL_DESCS.t,
   |   O_FORMAL_DESCS.t
   };
|
O_GENERICS_LIST
 ::= ()(O_GENERICS_LIST.reversed = (O_GENERICS_LIST.tail));
   | (O_GENERICS_LIST O_GENERICS)
   |   O_GENERICS$2.tail={
   |       OGENERICS.t::
   |       O_GENERICS_LIST$1.tail);
   | O_GENERICS_LIST$1.reversed=O_GENERICS_LIST$2.reversed;
   |
O_GENERICS
 ::= (GENERICS TYPE_DECLARATIONS REQMTS_TRACE)
   | (TYPE_DECLARATIONS.tail=TypeDeclNil;
   |   O_GENERICS.t = OpGenerics{
   |       TYPE_DECLARATIONS.reversed,
   |       REQMTS_TRACE.t
   };
   |
O_INPUTS_LIST
 ::= ()(O_INPUTS_LIST.reversed = (O_INPUTS_LIST.tail));
   | (O_INPUTS_LIST O_INPUTS)
   |   O_INPUTS$2.tail={
   |       O_INPUTS.t::
   |       O_INPUTS_LIST$1.tail);
   | O_INPUTS_LIST$1.reversed=O_INPUTS_LIST$2.reversed;
   |
O_INPUTS
 ::= (INPUTS TYPE_DECLARATIONS REQMTS_TRACE)
   | (TYPE_DECLARATIONS.tail=TypeDeclNil;
   |   O_INPUTS.t = OpInputs{
   |       TYPE_DECLARATIONS.reversed,
   |       REQMTS_TRACE.t
   };
   |
O_OUTPUTS_LIST
 ::= ()(O_OUTPUTS_LIST.reversed = (O_OUTPUTS_LIST.tail));
   | (O_OUTPUTS_LIST O_OUTPUTS)
   |   O_OUTPUTS$2.tail={
   |       O_OUTPUTS.t::
   |       O_OUTPUTS_LIST$1.tail);
   | O_OUTPUTS_LIST$1.reversed=O_OUTPUTS_LIST$2.reversed;
   |
O_OUTPUTS
::= (OUTPUTS TYPE_DECLARATIONS REQMTS_TRACE)
   (TYPE_DECLARATIONS.tail=TypeDeclNil;
    O_OUTPUTS.t = OpOutputs(
      TYPE_DECLARATIONS.reversed,
      REQMTS_TRACE.t
    ));

O_EXCEPTIONS_LIST
::= ()(O_EXCEPTIONS_LIST.reversed = (O_EXCEPTIONS_LIST.tail));
   (O_EXCEPTIONS_LIST$2.tail=
    O_EXCEPTIONS.t::
    O_EXCEPTIONS_LIST$1.tail);
   O_EXCEPTIONS_LIST$1.reversed=O_EXCEPTIONS_LIST$2.reversed;

O_EXCEPTIONS
::= (EXCEPTIONSW ALONE_ID_LIST REQMTS_TRACE)
   (ALONE_ID_LIST.tail = AIdNil;
    O_EXCEPTIONS.t = OpExceptions{
      ALONE_ID_LIST.reversed,
      REQMTS_TRACE.t
    });

O_TIMING_INFO
::= ()
   (O_TIMING_INFO.t = (OpTimingInfoNone));
   (MAXEXTIMEKW TIME REQMTS_TRACE)
   (O_TIMING_INFO.t = OpTimingInfo(
     TIME.t,
     REQMTS_TRACE.t
   ));

TIME::= (INTEGER TIME_UNIT)
   (TIME.t = Time(INTEGER.t, TIME_UNIT.t));

TIME_UNIT
::= [MICRO]
   (TIME_UNIT.t = UnitMICROSECONDS;
    [MS] (TIME_UNIT.t = UnitMS;
    [SEC] (TIME_UNIT.t = UnitSEC;
    [MIN] (TIME_UNIT.t = UnitMIN;
    [HOURS] (TIME_UNIT.t = UnitHOURS;

O_STATES_LIST
::= ()(O_STATES_LIST.reversed = (O_STATES_LIST.tail));
   (O_STATES_LIST 0_STATES)
   (O_STATES_LIST$2.tail={
     O_STATES.t::

O_STATES
::= (STATESW TYPE_DECLARATIONS INITIALKW EXPRESSION_LIST REQMTS_TRACE)
   (EXPRESSSION_LIST.tail=InitialExpListNil;
    TYPE_DECLARATIONS.tail=TypeDecNil;
    O_STATES.t = OpStates(
      TYPE_DECLARATIONS.reversed,
      EXPRESSION_LIST.reversed,
      REQMTS_TRACE.t
    ));

AN_ARGUMENT
::= (" EXPRESSION_LIST ")
   (EXPRESSSION_LIST.tail = InitialExpListNil;
    AN_ARGUMENT.t = AnArgument{EXPRESSSION_LIST.reversed});

INITIAL_ARGS
::= ()(INITIALARGS.reversed=INITIALARGS.tail);
APPENDIX G - Concrete Rules

(EXPR$1.t=ParenthesizedExp(EXPR$2.t));

/* BOOLEAN_EXPRESSION */
| NOTKW EXPRESSION prec NOTKW |
| EXPRESSION$1.t= |
| NotExp(EXPR$2.t); |
| (FALSEKW) |
| EXPRESSION$t=FALSE; |
| (TRUEKW) |
| EXPRESSION$t=TRUE; |

| EXPRESSION t EXPRESSION |
| EXPRESSION$1.t= |
| EqualExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

| EXPRESSION t EXPRESSION prec ']=' |
| EXPRESSION$1.t= |
| LessExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

| EXPRESSION t EXPRESSION prec '>=' |
| EXPRESSION$1.t= |
| GreaterEqualExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

| EXPRESSION GTEKW EXPRESSION prec GTEKW |
| EXPRESSION$1.t= |
| GreaterEqualExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

| EXPRESSION LTEKW EXPRESSION prec LTEKW |
| EXPRESSION$1.t= |
| LessEqualExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

| EXPRESSION NEQKW EXPRESSION prec NEQKW |
| EXPRESSION$1.t= |
| NotEqualExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

| EXPRESSION ANDKW EXPRESSION prec ANDKW |
| EXPRESSION$1.t= |
| AndExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

| EXPRESSION ORKW EXPRESSION prec ORKW |
| EXPRESSION$1.t= |
| OrExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

| EXPRESSION XORKW EXPRESSION prec XORKW |
| EXPRESSION$1.t= |
| XorExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

/* ARITHMETIC_EXPRESSION */
| INTEGER |
| EXPRESSION.t=Real( |
| INTEGER$1.t, |
| INTEGER$2.t |
| ); |

| EXPRESSION t EXPRESSION prec '+' |
| EXPRESSION$1.t= |
| PlusExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

| EXPRESSION t EXPRESSION prec '-' |
| EXPRESSION$1.t= |
| MinusExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

| EXPRESSION t EXPRESSION prec '*' |
| EXPRESSION$1.t= |
| TimesExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

| EXPRESSION t EXPRESSION prec '/' |
| EXPRESSION$1.t= |
| DivExp( |
| EXPRESSION$2.t, |
| EXPRESSION$3.t |
| ); |

| EXPRESSION t EXPRESSION prec '-' |
| EXPRESSION$1.t= |
| NegativeExp( |
| EXPRESSION$2.t |
| ); |

| EXPRESSION t EXPRESSION prec '+' |
| EXPRESSION$1.t= |
| PositiveExp( |
| EXPRESSION$2.t |
| ); |

| ABSKW ('EXPRESSION') prec ABSKW |
| EXPRESSION$1.t= |
APPENDIX G - Concrete Rules

```plaintext
AbsExp(
  EXPRESSION$2.t
);

I ( EXPRESSION MOD EXP EXPRESSION prec MODXW)
  EXPRESSION$1.t =
    ModExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
    );

I ( EXPRESSION REM EXP EXPRESSION prec REMRX)
  EXPRESSION$1.t =
    RemExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
    );

I ( EXPRESSION EXP EXP EXPRESSION prec EXPWM)
  EXPRESSION$1.t =
    ExponentExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
    );

/* string expression */
I ( EXPRESSION ' & ' EXPRESSION prec '&')
  EXPRESSION$1.t =
    ConcatExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
    );

O_KEYWEYS
  ::= [1](O_KEYWEYS.t=(KeywordsNone));

I ( KEYWOW ALONE_ID_LIST)
  (ALONE_ID_LIST.tail=Alt11);
  O_KEYWEYS.t=(Keywords(ALONE_ID_LIST.reversed));

O_INFORMAL_DESCS
  ::= [1](O_INFORMAL_DESCS.t=(InformalDescsNone));

I ( DESCRIPT W LCURLY yCommentLines RCURLY)
  { O_INFORMAL_DESCS.t = InformalDescs(yCommentLines.a); }

O_FORMAL_DESCS
  ::= [1](O_FORMAL_DESCS.t=(FormalDescsNone));

I ( AXIOMW LCURLY yCommentLines RCURLY)
  { O_FORMAL_DESCS.t = FormalDescs(yCommentLines.a); }

REQMTS_TRACE
  ::= [1](REQMTS_TRACE.t=(ReqmtsTraceNone));

I ( REQBYW ALONE_ID_LIST)
  (ALONE_ID_LIST.tail=Alt11);
  REQMTS_TRACE.t=ReqmtsTrace(ALONE_ID_LIST.reversed);

OPERATOR_IMPL
  ::= [1]({OPERATOR_IMPL$1.tail=OPERATOR_IMPL$2.tail};

I (OPERATOR_IMPL$1.T_O_P_IMPL)
  (OPERATOR_IMPL$1.LIST$2.tail=;
   T_O_P_IMPL.t::
     OPERATOR_IMPL$1.LIST$1.tail);
  OPERATOR_IMPL$1.LIST$2.tail=OPERATOR_IMPL$2.t.

T_O_P_IMPL
  ::= [1](OPERATORW ID OPERATOR_IMPL)
  (T_O_P_IMPL.t=TopImpl(
    ID.t,
    OPERATOR_IMPL.t
  ));

GRAPH::= (GRAPHW VERTEX LIST EDGE LIST)
  (EDGE LIST.tail=EdgeListNil);
  VERTEX LIST.tail=VertexListNil;
  GRAPH.t=(Graph(
    VERTEX LIST.reversed,
    EDGE LIST.reversed)
  ));

VERTEX LIST
  ::= [1](VERTEX LIST.reversed=VERTEX LIST.tail);

I (VERTEX LIST A_VERTEX)
  { VERTEX LIST$2.tail=(A_VERTEX.t::
         VERTEX LIST$2.tail=
         A_VERTEX t::}
```

262
APPENDIX G - Concrete Rules

```plaintext
VERTEX_LIST$1.tail

VERTEX_LIST$1.reversed=VERTEX_LIST$2.reversed;
}

A_VERTEX
  ::= (VERTEX$9 OPERATOR_ID OPTIONAL_TIME)
       {A_VERTEX.t=Vertex(
         OPERATOR_ID.t,
         OPTIONAL_TIME.t
       )};

OPTIONAL_TYPE_ID
  ::= ()
       {OPTIONAL_TYPE_ID.t = OptionalTypeIdNull;}
     | {ID}
       {OPTIONAL_TYPE_ID.t = OptionalTypeId(ID.t);};

OPERATOR_ID
  ::= (ID OPERATOR_ID_PAIRS)
       {OPERATOR_ID.t=OperatorId(OptionalTypeIdNull, ID.t, OPERATOR_ID_PAIRS.t)}
     | (OPTIONAL_TYPE_ID , ID OPERATOR_ID_PAIRS)
       {OPERATOR_ID.t=OperatorId(OPTIONAL_TYPE_ID.t, ID.t, OPERATOR_ID_PAIRS.t)};

OPERATOR_ID_PAIRS
  ::= ()(OPERATOR_ID_PAIRS.t=(OperatorIdPairsNull))
     | (', ' ALONE_ID_LIST ' ' ALONE_ID_LIST ' ')
       {ALONE_ID_LIST$1.tail = AInNil;
        ALONE_ID_LIST$2.tail = AInNil;
        OPERATOR_ID_PAIRS.t=OperatorIdPairs(
          ALONE_ID_LIST$1.reversed,
          ALONE_ID_LIST$2.reversed )};

OPTIONAL_TIME
  ::= ()(OPTIONAL_TIME.t=(OptionalTimeNull))
     | (', ' TIME)
       {OPTIONAL_TIME.t=OptionalTime(
        TIME.t )};

LATENCY_TIME
  ::= ()($$.t = (LatencyTimeNull))
     | (', ' TIME)
       {LATENCY_TIME.t = LatencyTime(TIME.t)};

EDGE_LIST
  ::= ()
       {EDGE_LIST.reversed=EDGE_LIST.tail;}
     | (EDGE_LIST AN_EDGE)
       {EDGE_LIST$2.tail=(
         AN_EDGE.t::
         EDGE_LIST$1.tail
       )};
     | EDGE_LIST$1.reversed=EDGE_LIST$2.reversed;

AN_EDGE
  ::= (EDGE$9 ID LATENCY_TIME FROM_VERTEX_ID ARROW TO_VERTEX_ID)
       {AN_EDGE.t=AnEdge(
        ID.t,
        LATENCY_TIME.t,
        FROM_VERTEX_ID.t,
        TO_VERTEX_ID.t )};

FROM_VERTEX_ID
  ::= |ID OPERATOR_ID_PAIRS)
       {$$.t = VertexId(OptionalTypeIdNull, ID.t, OPERATOR_ID_PAIRS.t)}
     | (OPTIONAL_TYPE_ID , ID OPERATOR_ID_PAIRS)
       {$$.t = VertexId(
         OPTIONAL_TYPE_ID.t, ID.t, OPERATOR_ID_PAIRS.t)};

TO_VERTEX_ID
  ::= |ID OPERATOR_ID_PAIRS)
       {$$.t = VertexId(OptionalTypeIdNull, ID.t, OPERATOR_ID_PAIRS.t)}
     | (OPTIONAL_TYPE_ID , ID OPERATOR_ID_PAIRS)
       {$$.t = VertexId(
         OPTIONAL_TYPE_ID.t, ID.t, OPERATOR_ID_PAIRS.t)};

DECLARATIONS
  ::= (DECLARATIONS.t=Declarations(
         OPTIONAL_STREAMS, OPTIONAL_TIMERS)
     | OPTIONAL_STREAMS
     | OPTIONAL_TIMERS)
       {DECLARATIONS.t=Declarations(
         OPTIONAL_STREAMS.t,
         OPTIONAL_TIMERS.t )};
```

263
APPENDIX G - Concrete Rules

OPTIONAL_TRIGGER ::= () | {OPTIONAL_TRIGGER.t = (OptionalTriggerNull);}

| (TRIGGERBYKW
  TYPE_OF_TRIGGER
  ALONE_ID_LIST
  OPTIONAL_IF_PREDICATE
  REQMTS_TRACE)
  (ALONE_ID_LIST.tail = ALoneNil;
  OPTIONAL_TIMERS.t = Timers{
    ALONE_ID_LIST.reversed;
  });

| (TRIGGERBYKW
  IPKW
  EXPRESSION
  REQMTS_TRACE)
  (OPTIONAL_TRIGGER.t = OptionalIfExp{
    EXPRESSION.t,
    REQMTS_TRACE.t
  });

TYPE_OF_TRIGGER ::= [ALLKW] (TYPE_OF_TRIGGER.t = TriggerAll());
| [SOMEKW] (TYPE_OF_TRIGGER.t = TriggerSome());

OPTIONAL_PERIOD ::= () | {OPTIONAL_PERIOD.t = (OptPeriodNull);}

| (PERKW TIME REQMTS_TRACE)
  (OPTIONAL_PERIOD.t = OptPeriod{
    TIME.t,
    REQMTS_TRACE.t
  });

OPTIONAL_FINISH_WITHIN ::= () | {OPTIONAL_FINISH_WITHIN.t = (OptFinishWithinNull);}

| (FINISHKW TIME REQMTS_TRACE)
  (OPTIONAL_FINISH_WITHIN.t = OptFinishWithin{
    TIME.t,
    REQMTS_TRACE.t
  });

OPTIONAL_MCP ::= () | {OPTIONAL_MCP.t = (OptMcpNull);}

| (MCPKW TIME REQMTS_TRACE)
  (OPTIONAL_MCP.t = OptMcp{
    TIME.t,
    REQMTS_TRACE.t
  });

OPTIONAL_HRT ::= () | {OPTIONAL_HRT.t = (OptHrtNull);}
APPENDIX G - Concrete Rules

```plaintext
| (HRTKW TIME REQMTS_TRACE)
| (OPTIONAL_HRT t=OptHrt{
| TIME.t,
| REQMTS_TRACE.t
| });
|
| OUTPUT_GUARDS
|::= ()(OUTPUT_GUARDS.reversed=OUTPUT_GUARDS.tail);
| (OUTPUT_GUARDS A_GUARD)
| (OUTPUT_GUARDS$2.tail={
| A_GUARD.t::
| OUTPUT_GUARDS$1.tail
| });
| OUTPUT_GUARDS$1.reversed=OUTPUT_GUARDS$2.reversed;
|
| A_GUARD:= (OPTIONAL A_LIST IFKW C_EXPRESSION REQMTS_TRACE)
| (A_LIST.tail = A1Nil;
| A_GUARD t=AGuard(
| A_LIST.tail.reversed,
| C_EXPRESSION.t,
| REQMTS_TRACE.t
| ));
|
| EXCEPTION_OPS
|::= ()(EXCEPTION_OPS.t=ExceptionOpsNull);
| (EXCEPTION_OPTIONS)
| (EXCEPTION_OPTIONS.tail=ExceptionOptionsNil):
| EXCEPTION_OPS.t=Exception(
| EXCEPTION_OPTIONS.reversed
| );
|
| EXCEPTION_OPTIONS
|::= AN_EXCEPTION
| (EXCEPTION_OPTIONS.reversed={
| AN_EXCEPTION.t::
| EXCEPTION_OPTIONS.tail
| });
| (EXCEPTION_OPTIONS AN_EXCEPTION)
| (EXCEPTION_OPTIONS$2.tail={
| AN_EXCEPTION.t::
| EXCEPTION_OPTIONS$1.tail
| });
| EXCEPTION_OPTIONS$1.reversed=EXCEPTION_OPTIONS$2.reversed;
|
| AN_EXCEPTION
|::= (OPTIONAL ID OPTIONAL_IF_PREDICATE REQMTS_TRACE)
| (AN_EXCEPTION.t=AnException(
| ID.t,
| OPTIONAL_IF_PREDICATE.t,
| REQMTS_TRACE.t
| ));
|
| TIMER_OPERATIONS
|::= ()(TIMER_OPERATIONS.reversed=TIMER_OPERATIONS.tail);
| (TIMER_OPERATIONS A_TIMER_OPERATION)
| (TIMER_OPERATIONS$2.tail={
| A_TIMER_OPERATION.t::
| TIMER_OPERATIONS$1.tail
| });
| TIMER_OPERATIONS$1.reversed=TIMER_OPERATIONS$2.reversed;
|
| A_TIMER_OPERATION
|::= (RESETKW ID OPTIONAL_IF_PREDICATE REQMTS_TRACE)
| (A_TIMER_OPERATION.t=ATimerReset{
| ID.t,
| OPTIONAL_IF_PREDICATE.t,
| REQMTS_TRACE.t
| });
| (STOPKW ID OPTIONAL_IF_PREDICATE REQMTS_TRACE)
| (A_TIMER_OPERATION=tATimerStop{
| ID.t,
| OPTIONAL_IF_PREDICATE.t,
| REQMTS_TRACE.t
| });
| (STARTKW ID OPTIONAL_IF_PREDICATE REQMTS_TRACE)
| (A_TIMER_OPERATION=tATimerStart{
| ID.t,
| OPTIONAL_IF_PREDICATE.t,
| REQMTS_TRACE.t
| });
|
| OPTIONAL_IF_PREDICATE
|::= ()(OPTIONAL_IF_PREDICATE.t=OptIfPredicateNil);
| (IFKW C_EXPRESSION)
| (OPTIONAL_IF_PREDICATE.t=IfPredicate( C_EXPRESSION)
| });
|
| C_AN_ARGUMENT
|::= ("C_EXPRESSION_LIST ")
| (C_EXPRESSION_LIST.tail = CInitialExpListNil;
| C_AN_ARGUMENT t=CanArgument (C_EXPRESSION_LIST.reversed)
| );
|
| C_INITIAL_ARGS
|::= ()(C_INITIAL_ARGS.reversed={C_INITIAL_ARGS.tail});
| (C_INITIAL_ARGS C_AN_ARGUMENT)
| (C_INITIAL_ARGS$2.tail={
| C_AN_ARGUMENT.t::C_INITIAL_ARGS$1.tail
| });
| C_INITIAL_ARGS$1.reversed=C_INITIAL_ARGS$2.reversed;
|
| C_EXPRESSION_LIST
|::= (C_EXPRESSION)
| (C_EXPRESSION_LIST.reversed=}
```
APPENDIX G - Concrete Rules

(C_EXPRESSION t::C_EXPRESSION_LIST.tail);

(C_EXPRESSION_LIST t::null)
(C_EXPRESSION t::C_EXPRESSION_LIST2.tail)
(C_EXPRESSION_LIST t::C_EXPRESSION_LIST1.tail)

C_EXPRESSION t::C_EXPRESSION_LIST1 reversed=C_EXPRESSION_LIST2 reversed;

C_EXPRESSION t::C_EXPRESSION t::Identifier(ID t::)

{QUOTEW yCommentLines QUOTEW}
(C_EXPRESSION t::C_Textual_Description yCommentLines.a::)

{TYPE_NAME '.' ID C_INITIAL_ARGS}

C_EXPRESSION t::C_TypeExpression

TYPE_NAME t::

{C_INITIAL_ARGS.reversed}

)

{('C_EXPRESSION')}

C_EXPRESSION t::C_ParenthesizedExp(C_EXPRESSION t::)

{TIME}

C_EXPRESSION t::C_TinaExpression(TIME t::)

/* BOOLEAN_EXPRESSION */

{NOTKW C_EXPRESSION prec NOTKW}

C_EXPRESSION t::

C_NotExp(C_EXPRESSION t::)

{FALSEKW}

C_EXPRESSION t::C_False;

{TRUEKW}

C_EXPRESSION t::C_True;

{C_EXPRESSION '=' C_EXPRESSION prec '='}

C_EXPRESSION t::

C_EqualsExp(C_EXPRESSION t::

C_EXPRESSION t::)

{C_EXPRESSION '<' C_EXPRESSION prec '<'}

C_EXPRESSION t::

C_LessExp(C_EXPRESSION t::

C_EXPRESSION t::)

{C_EXPRESSION '>' C_EXPRESSION prec '>'}

C_EXPRESSION t::

C_GreaterExp(C_EXPRESSION t::

C_EXPRESSION t::)

{C_EXPRESSION GTEKW C_EXPRESSION prec GTEKW}

C_EXPRESSION t::

C_GreatEqualExp(C_EXPRESSION t::

C_EXPRESSION t::)

{C_EXPRESSION LTEKW C_EXPRESSION prec LTEKW}

C_EXPRESSION t::

C_LessEqualExp(C_EXPRESSION t::

C_EXPRESSION t::)

{C_EXPRESSION NEQKW C_EXPRESSION prec NEQKW}

C_EXPRESSION t::

C_NotEqualExp(C_EXPRESSION t::

C_EXPRESSION t::)

{C_EXPRESSION ANDKW C_EXPRESSION prec ANDKW}

C_EXPRESSION t::

C_AndExp(C_EXPRESSION t::

C_EXPRESSION t::)

{C_EXPRESSION ORKW C_EXPRESSION prec ORKW}

C_EXPRESSION t::

C_OrExp(C_EXPRESSION t::

C_EXPRESSION t::)

{C_EXPRESSION XORKW C_EXPRESSION prec XORKW}

C_EXPRESSION t::

C_XorExp(C_EXPRESSION t::

C_EXPRESSION t::)

/* ARITHMETIC_EXPRESSION */

{INTEGER}

C_EXPRESSION t::C_Integer(INTEGER t::)

{INTEGER '.' INTEGER}

C_EXPRESSION t::C_Real(INTEGER t::

INTEGER t::

INTEGER t::)

{C_EXPRESSION '*' C_EXPRESSION prec '*'}

C_EXPRESSION t::

C_MulExp(C_EXPRESSION t::

C_EXPRESSION t::)

{C_EXPRESSION '/' C_EXPRESSION prec '/'}

C_EXPRESSION t::

C_DivExp(C_EXPRESSION t::

C_EXPRESSION t::)

{C_EXPRESSION '-' C_EXPRESSION prec '-'}

C_EXPRESSION t::

CMinusExp(C_EXPRESSION t::

C_EXPRESSION t::)
APPENDIX G - Concrete Rules

C_EXPRESSION$3.t

{(C_EXPRESSION """ C_EXPRESSION prec """")
 (C_EXPRESSION$1.t=
 CTimesExp{
 C_EXPRESSION$2.t,
 C_EXPRESSION$3.t
 });
}

{(C_EXPRESSION "/" C_EXPRESSION prec "/")
 (C_EXPRESSION$1.t=
 CDivExp{
 C_EXPRESSION$2.t,
 C_EXPRESSION$3.t
 });
}

{('=' C_EXPRESSION prec '++')
 (C_EXPRESSION$1.t=
 CPositiveExp{
 C_EXPRESSION$2.t
 });
}

{('-' C_EXPRESSION prec '-')
 (C_EXPRESSION$1.t=
 CNegativeExp{
 C_EXPRESSION$2.t
 });
}

{ABSNW ('(C_EXPRESSION')' prec ABSNW)
 (C_EXPRESSION$1.t=
 CABbExp{
 C_EXPRESSION$2.t
 });
}

{(C_EXPRESSION MODNW C_EXPRESSION prec MODNW)
 (C_EXPRESSION$1.t=
 CModExp{
 C_EXPRESSION$2.t,
 C_EXPRESSION$3.t
 });
}

{(C_EXPRESSION REMNW C_EXPRESSION prec REMNW)
 (C_EXPRESSION$1.t=
 CRemExp{
 C_EXPRESSION$2.t,
 C_EXPRESSION$3.t
 });
}

{(C_EXPRESSION EXPNW C_EXPRESSION prec EXPNW)
 (C_EXPRESSION$1.t=
 CExpnentExp{
 C_EXPRESSION$2.t,
 C_EXPRESSION$3.t
 });
}

/* STRING_EXPRESSION */

{(C_EXPRESSION '"' C_EXPRESSION prec '"')
 (C_EXPRESSION$1.t=
 CConcatExp{
 C_EXPRESSION$2.t,
 C_EXPRESSION$3.t
 });
}
<table>
<thead>
<tr>
<th>No.</th>
<th>Initial Distribution List</th>
</tr>
</thead>
</table>
| 1.  | Defense Technical Information Center  
     8725 John J. Kingman Road, Suite 0944  
     Fort Belvoir, VA  22060 | 2 |
| 2.  | Dudley Knox Library  
     Naval Postgraduate School  
     411 Dyer Road  
     Monterey, CA  93943 | 2 |
| 3.  | Center for Naval Analysis  
     4401 Ford Avenue  
     Alexandria, VA  22302 | 1 |
| 4.  | Dr. Ted Lewis, Chairman, Code CS/L  
     Computer Science Department  
     Naval Postgraduate School  
     Monterey, CA  93943 | 1 |
| 5.  | Chief of Naval Research  
     800 North Quincy Street  
     Arlington, VA  22217 | 1 |
| 6.  | Dr. Luqi, Code CS/Lq  
     Computer Science Department  
     Naval Postgraduate School  
     Monterey, CA  93943 | 18 |
| 7.  | Capt Scott Groshenheider  
     125 6th AVE S.W.  
     LeMars, IA  51031 | 3 |
| 8.  | The Groshenheider’s  
     214 1st Street S.W.  
     LeMars, IA  51031 | 1 |
| 9.  | The Betsworth’s  
     125 6th AVE S.W.  
     LeMars, IA  51031 | 1 |
| 10. | Ada Joint Program Office  
      OUSDRE (R&AT)  
      The Pentagon  
      Washington, DC  20301 | 1 |